



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise Comparativa de Projeto e Administração de Banco de Dados Entre os SGBDs Cassandra e MySQL

Hermano Portella Leite
Igor da Silva Bonomo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof. Dr. Maristela Terto de Holanda

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Rodrigo Bonifácio

Banca examinadora composta por:

Prof. Dr. Maristela Terto de Holanda (Orientadora) — CIC/UnB

Prof. Henrique Pereira de Freitas Filho — DAA/IFG

Prof. Ruben Cruz Huacarpuma — CIC/UnB

CIP — Catalogação Internacional na Publicação

Leite, Hermano Portella.

Análise Comparativa de Projeto e Administração de Banco de Dados
Entre os SGBDs Cassandra e MySQL / Hermano Portella Leite, Igor
da Silva Bonomo. Brasília : UnB, 2016.

117 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. Cassandra, 2. DBA, 3. NoSQL, 4. SGBD

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília-DF — Brasil



Análise Comparativa de Projeto e Administração de Banco de Dados Entre os SGBDs Cassandra e MySQL

Hermano Portella Leite
Igor da Silva Bonomo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof. Henrique Pereira de Freitas Filho Prof. Ruben Cruz Huacarpuma
DAA/IFG CIC/U n B

Prof. Dr. Rodrigo Bonifácio
Coordenador do Bacharelado em Ciência da Computação

Brasília, 16 de agosto de 2016

Dedicatória

Dedico a minha família que sempre acreditou em mim, me dando suporte e motivação para continuar seguindo meus sonhos(Hermano).

Dedico a três pessoas muito importantes na minha vida, a primeira meu avô Orly Bonomo quem sempre acreditou em mim, me ensinou tudo desde pequeno, sempre esteve do meu lado, me apoiando, me ajudando e me aconselhando com tudo, foi de extrema importância para mim, ele não pode estar presente para ver eu concluindo meu curso, mas nunca será esquecido por mim e por aqueles que o amam. Obrigado Vô por tudo. A segunda é meu avô Manoel Paula da Silva que também sempre acreditou em mim, me passando seus conhecimentos e ensinamentos, e que mesmo não estando perto de mim todos os dias, me apoiou e me ajudou sempre, e que também não pode estar presente para ver para ver eu concluindo meu curso, mas nunca será esquecido. Obrigado Vô por tudo. A terceira pessoa é meu sobrinho Luan Victor Bonomo que nasceu em agosto de 2016, que mesmo com pouco tempo de vida eu já o amo de um jeito, que eu quero deixar para ele todo meu conhecimento, ensinamentos e estar sempre ao seu lado para tudo que ele precisar assim como meus avôs, obrigado Luan(Igor).

Agradecimentos

Agradeço minha família que é a base para tudo, e sem eles nada seria possível. Meus amigos, que fizeram parte de toda graduação, tornando ela mais fácil e divertida. E aos professores, que nos transmitiram todo seu conhecimento da melhor maneira possível, especialmente a professora Maristela que nos ajudou a finalizar a graduação com este trabalho final(Hermano).

Agradeço a minha família, minha mãe Leidiane, meu pai Gibson, meu irmão Victor, meu sobrinho Luan, minha vó Maria das Neves, meu avô Orly, meu avô Paulo, minha tia Symone, meu tio Luciano, minha prima Duda, meu primo Júlio, meus tios Waldir e Leila, minhas tias Lilian e Liliane, que são a minha base, meu tudo, que sem eles eu não conseguiria nada. Meus amigos, que estiveram ao meu lado durante todos os momentos, que me apoiaram, me ajudaram e passamos bons momentos juntos para aguentar nos momentos mais difíceis da graduação. E aos professores, principalmente a Maristela pela ajuda e esforço com este trabalho final(Igor).

Resumo

O Administrador de Banco de Dados (DBA) é o profissional responsável por gerenciar e cuidar dos dados de uma aplicação, utilizando um Sistema Gerenciador de Banco de Dados (SGBD), e o projetista de banco de dados é aquele responsável pela modelagem deste BD. Com a evolução das aplicações Web, a forma como os dados são gerenciados sofreram grandes alterações, o que consequentemente influencia as atividades realizadas tanto pelos DBAs, quanto pelos projetistas. O Cassandra é um SGBD NoSQL que busca facilitar a escalabilidade, garantindo uma disponibilidade contínua com tolerância a falhas. Este possui arquitetura, modelagem e estrutura diferentes do proposto pelo modelo relacional. Este trabalho tem como objetivo realizar uma análise comparativa das atividades do DBA e do projetista no Cassandra e no modelo relacional, utilizando o SGBD MySQL.

Palavras-chave: Cassandra, DBA, NoSQL, SGBD

Abstract

Database administrator (DBA) is a professional responsible for taking care and managing the data of an application, using for this purpose a Database Management System (DBMS) and Data modeler is the one responsible of database modeling. With the evolution of web applications, the way data is managed has suffered major changes, which impacted on DBA's and Data modelers activities. Cassandra is a NoSQL DBMS that makes it easy to scale, ensuring continuous availability with fault tolerance. It also has a different architecture, data model and structure when compared to Relational Databases. This study has the goal of making a comparative analysis of DBA's and Data modelers activities using Cassandra and Relation Model DBMS, using MySQL.

Keywords: Cassandra, DBA, NoSQL, DBMS

Sumário

1	Introdução	1
1.1	Problema e Hipótese	2
1.2	Objetivo	2
1.2.1	Objetivo Geral	2
1.2.2	Objetivo Específico	3
1.3	Estrutura do Trabalho	3
2	Banco de Dados	4
2.1	Sistema Gerenciador de Bancos de Dados	4
2.2	O Modelo Relacional	5
2.2.1	Formas Normais	5
2.2.2	Transações	6
2.2.3	Linguagem de Consulta(SQL)	6
2.2.4	MySQL	7
2.3	Banco de Dados <i>NoSQL</i>	7
2.3.1	Características dos Bancos de Dados <i>NoSQL</i>	8
2.3.2	Modelo de Dados <i>NoSQL</i>	9
2.4	Projeto e Administração de Banco de Dados	11
2.4.1	Responsabilidades e Tarefas de um DBA	12
2.4.2	Responsabilidades e Tarefas de um Projetista	13
2.4.3	Tipos de DBA	14
3	NoSQL Cassandra	15
3.1	O Cassandra	15
3.2	Arquitetura	16
3.2.1	Estruturas Básicas Principais	17
3.2.2	Princípios importantes	18
3.3	Modelo de Dados	23
3.3.1	<i>Cluster</i>	23
3.3.2	<i>Keyspace</i>	23
3.3.3	Família de Colunas	24
3.3.4	Coluna	24
3.4	CQL	25

4	Resultados	26
4.1	Contextualização	26
4.2	Análise Comparativa entre os SGBDs	28
4.2.1	Modelo de Dados	28
4.2.2	Segurança	39
4.2.3	Backup e Recuperação	43
4.2.4	Tabela comparativa	45
5	Conclusão	47
	Referências	49

Lista de Figuras

2.1	Teorema CAP.	8
3.1	Cassandra: Escalabilidade Linear [1].	16
3.2	Arquitetura Cassandra.	17
3.3	Exemplo hashing consistente.	20
3.4	Arquitetura Vnode vs token único.	21
3.5	Modelo de Dados.	25
4.1	Imagem cluster utilizado.	27
4.2	Cluster criado.	28
4.3	Diagrama Entidade Relacionamento para o cenário proposto.	31
4.4	Modelo lógico para o MySQL.	32
4.5	Modelo lógico para o Cassandra demonstrando todas as regras sendo aplicadas.	34
4.6	Modelo lógico completo utilizando a notação Chebotko.	34
4.7	Modelo físico Cassandra.	35
4.8	Comando CQL para criação de tabela.	35
4.9	Comando SQL para criação de tabela.	36
4.10	Importação de dados do MySQL para o Cassandra.	36
4.11	Consulta em SQL para gerar arquivo CSV.	37
4.12	Comando para copiar dados do arquivo CSV para o CQL.	37
4.13	Consulta SQL.	38
4.14	Consulta CQL.	38
4.15	Criação de usuário em SQL.	40
4.16	Criação de usuário em CQL.	40
4.17	Criação de usuário em SQL com host.	41
4.18	Utilização do <i>role</i> no Cassandra.	42
4.19	Comando Backup MySQL.	44
4.20	Comando Backup Cassandra.	45

Lista de Tabelas

4.1	Tabela privilégios MySQL.	42
4.2	Tabela comparativa das atividades para os SGBDs Cassandra e MySQL. .	46

Capítulo 1

Introdução

A evolução da tecnologia, bem como a utilização atual da Internet estão mudando a forma como os dados são tratados em torno de três principais pontos: volume, velocidade e variedade [17]. O aumento no volume de dados pode ser observado como mostra a Cisco, conceituada empresa americana da área de redes e comunicação, que mostra que o tráfego global na internet em 2016 chega a ultrapassar um *zettabyte* por ano [3]. Além disso, diversas aplicações trabalham com dados em tempo real o que exige uma grande velocidade na forma como esses são tratados. Estes dados ainda podem possuir forma de mensagem, imagem, sinal de GPS de celular, vídeo, entre outras, que aumentam a variedade de dados que devem ser administrados.

Desde a década de 70, os bancos de dados são utilizados para organizar e relacionar os dados, principalmente por meio do modelo relacional. O sistema gerenciador de banco de dados tem um papel fundamental nesta área por ser responsável pela administração destes bancos de dados, garantindo a consistência dos dados quando mais de um usuário utiliza os mesmos simultaneamente, oferecendo uma interface amigável para realizar atividades como *backup*, restauração, recuperação de dados, e gerenciando uma grande quantidade de dados. O Sistema Gerenciador de Banco de Dados (SGBD) é um conjunto de ferramentas de *software* que controlam o acesso, organizam, armazenam, gerenciam, recuperam e mantêm dados em um banco de dados [24].

Um importante SGBD do modelo relacional é o MySQL que atualmente é desenvolvido e distribuído pela *Oracle Corporation*. Ele é distribuído em duas versões, a *Community server* que é de graça, e *Enterprise server* que é paga e oferece maiores benefícios. Assim como os outros SGBDs do modelo relacional, o MySQL também utiliza o SQL como forma de comunicação do usuário com o SGBD. Existem diversas interfaces gráficas que são utilizadas juntamente com o MySQL a fim de facilitar a interação do usuário. Algumas empresas que utilizam o MySQL são: Globo, Nasa, Youtube entre outras ¹.

Existe uma função específica dentro de uma empresa para administrar o banco de dados, a de administrador de banco de dados ou DBA (*Database Administrator*), a qual é responsável por garantir o correto funcionamento do SGBD, assim como sua contínua disponibilidade e segurança. Outra função importante é a do projetista de banco de dados que tem como uma de suas responsabilidades modelar os dados da empresa, buscando diminuir redundância e minimizar duplicidade, a fim de garantir um banco de dados consistente e eficiente.

¹Informação disponível em <https://www.mysql.com/customers/>

Tradicionalmente SGBDs relacionais tem sido utilizados para armazenamento de dados. Porém, ao longo do tempo, houve a necessidade de um modelo de dados mais flexível, que suportasse uma grande quantidade de dados chegando em alta velocidade e de diferentes dispositivos em tempo real e garantindo um bom desempenho. Assim, iniciou-se uma nova plataforma de gerenciamento de dados, os chamados NoSQL.

NoSQL, sigla utilizada para *Not only SQL*, que na tradução para o português significa "Não somente SQL", é o termo usado para expressar outras formas de se tratar dados, diferentes do modelo relacional. Os bancos NoSQL aparecem como alternativa para gestão de grande volume de dados, propondo alta escalabilidade, disponibilidade e desempenho. Essa nova forma se diferencia do modelo relacional principalmente porque não se prende tão fortemente às propriedades ACID, os quais são base para o modelo relacional [16].

O modelo NoSQL tem um grande impacto no trabalho realizado por um administrador de banco de dados. Mullins em [20] apresenta a necessidade de administração do NoSQL e que todo SGBD tem que ser configurado e gerenciado, sendo assim todo SGBD necessita de ser administrado.

Então, a proposta deste trabalho é comparar como a administração e a modelagem do banco de dados NoSQL se diferencia da administração e modelagem do banco de dados relacional, buscando assim, analisar as tarefas e responsabilidades de um administrador de banco de dados pontuando semelhanças e diferenças.

Atualmente existem mais de 225 SGBDs NoSQL², sendo que neste trabalho é utilizado o Apache Cassandra. Este SGBD é baseado no *BigTable* da *Google* e no *Dynamo* da *Amazon* mas foi desenvolvido dentro do *Facebook* e lançado em 2008. Atualmente é utilizado em mais de 400 empresas incluindo *Netflix*, *Spotify*, *Apple* e *Instagram* e em 39 países [26]. O Cassandra é um sistema de armazenamento distribuído escalável *open-source*, e é uma boa opção para gerenciamento de grande quantidade de dados não estruturados, semi-estruturados e estruturados sobre múltiplos centros de dados e na nuvem. Tem como característica uma alta disponibilidade com escalabilidade linear sem um único ponto de falha.

1.1 Problema e Hipótese

O problema foco deste trabalho é a realização de uma análise sobre as atividades de administração e modelagem do SGBD NoSQL Cassandra, buscando entender suas maiores dificuldades e facilidades, assim como suas vantagens e desvantagens sobre o modelo relacional. A hipótese do projeto é que a administração de banco de dados sofreu grandes mudanças com a utilização do Cassandra quando comparado com o modelo relacional.

1.2 Objetivo

1.2.1 Objetivo Geral

O objetivo geral desta pesquisa é realizar uma análise de sobre a administração e modelagem do SGBD Cassandra, observando as atividades do DBA e comparando-as com as suas atividades e com as realizadas em um SGBD relacional por meio do MySQL.

²Informação disponível em <http://nosql-database.org/>

1.2.2 Objetivo Específico

Para atingir esse propósito, foram definidos os seguintes objetivos específicos:

- Análise das estruturas do Modelo Relacional;
- Análise das estruturas do banco de dados NoSQL Cassandra;
- Análise das funções, atividades e responsabilidades dos Administradores e Projetistas de Bancos de Dados;
- Definição das funções do DBA;
- Aplicação do processo de Modelagem de Dados no Cassandra e no MySQL;
- Análise comparativa entre a administração dos SGBDs MySQL e Cassandra.

1.3 Estrutura do Trabalho

Este documento foi dividido nos capítulos apresentados a seguir:

- Capítulo 2: Realiza uma introdução a banco de dados, tratando inicialmente sobre os sistemas gerenciadores de banco de dados. Aborda-se também o Modelo Relacional explicando suas características, formas normais e propriedades ACID, assim como a linguagem SQL. Também se estuda o Modelo NoSQL, abordando suas características e tratando das diferentes classificações existentes. Por fim são abordados as figuras do administrador e do projetista de banco de dados, mostrando suas principais atividades, responsabilidades e tarefas. Aborda também os diferentes tipos de DBAs existentes;
- Capítulo 3: Apresenta o banco de dados NoSQL Cassandra, explicando suas principais características onde se busca entender o funcionamento geral deste SGBD. Também se estuda a arquitetura do Cassandra demonstrando suas estruturas e princípios importantes, como a comunicação entre os nós do SGBD, recuperação de falhas e como os dados são particionados. O modelo de dados do Cassandra também é exposto tratando conceitos como *cluster*, *keyspace*, família de colunas e colunas. Por fim se estuda suas operações internas e a linguagem CQL;
- Capítulo 4: Analisa as atividades administrativas nos SGBDs MySQL e Cassandra, realizando a comparação quanto a modelagem dos dados, explorando o modelo conceitual, lógico e físico. Compara também quanto a segurança, tratando como é realizada a autenticação e a autorização nos diferentes SGBDs. Demonstra as diferenças e semelhanças nas atividades backup e recuperação realizadas no Cassandra e no MySQL e por fim apresenta uma tabela comparativa resumindo as principais análises obtidas;
- Capítulo 5: Breve conclusão do trabalho realizado, recapitulando o que foi feito ao longo do estudo, com suas análises e resultados. Por fim, se expõe a possibilidade de trabalhos futuros.

Capítulo 2

Banco de Dados

Este capítulo aborda os conceitos sobre banco de dados relacional e NoSQL. Na seção 2.1 O Sistema Gerenciador de Banco de Dados é definido. A seção 2.2 explica sobre o modelo relacional. A seção 2.3 explica sobre os bancos de dados NoSQL. E por fim a seção 2.4 apresenta os conceitos de projeto e administração de banco de dados.

2.1 Sistema Gerenciador de Bancos de Dados

Banco de dados é uma coleção de dados relacionados. Esses dados são fatos que podem ser gravados e que possuem um significado implícito. O banco de dados possui as seguintes propriedades [11]:

- Um banco de dados representa alguns aspectos do mundo real, chamado de minimundo. As mudanças no minimundo são refletidas em um banco de dados;
- Um banco de dados é uma coleção lógica e coerente de dados com algum significado inerente;
- Um banco de dados é projetado, construído e povoado por dados, atendendo a uma proposta específica.

Um Sistema Gerenciador de Banco de Dados(SGBD) é uma coleção de programas que permite aos usuários criar e manter bancos de dados. Estes programas fazem com que o usuário consiga executar certos procedimentos para gerenciar banco de dados com uma maior facilidade. Desta forma, o SGBD é um sistema de *software* de propósito geral que facilita os processos que são: definição, construção, manipulação e compartilhamento de banco de dados entre vários usuários e aplicações [11], onde:

- Definição: significa especificar o que tem dentro de um Banco de Dados, ou seja, os tipos de dados, as estruturas e as restrições;
- Construção: remete ao processo de armazenamento de dados em um lugar específico do SGBD;
- Manipulação: implica nas funções que o SGBD executa, como pesquisar, atualizar e gerar relatórios em um Banco de Dados;
- Compartilhamento: permite que múltiplos usuários e programas acessem o Banco de Dados.

2.2 O Modelo Relacional

O Modelo Relacional foi criado no início da década de 70 proposto pelo Edgar Codd, pesquisador da IBM, com o intuito de representar os dados de uma forma mais simples, diminuindo a complexidade das consultas. Este modelo, com sua simplicidade e base matemática, atraiu a atenção rapidamente[23].

Ainda na década de 70 a IBM a qual já utilizava o modelo relacional criou uma linguagem de pesquisa declarativa chamada SQL (*Structured Query Language*) que se baseava bastante na álgebra relacional. Outras linguagens também foram criadas até que houve a necessidade da criação de um padrão, o que foi feito pela *American National Standards Institute* (ANSI)[24].

No modelo relacional, cada linha na tabela representa um fato que corresponde a uma entidade ou relacionamento do mundo real. Usa-se nomes para tabela e para as colunas de forma que ajudem na compreensão e interpretação do significado dos valores de cada linha. Na terminologia do modelo relacional formal, uma linha é chamada de tupla, um cabeçalho de coluna é conhecido como atributo e a tabela é chamada de relação, e o tipo de dados que descreve os valores é representado pelo domínio de valores possíveis. [11]

2.2.1 Formas Normais

O processo de normalização é uma série de testes, aplicados a um esquema de relação, a fim de verificar se este satisfaz uma certa forma normal. A normalização de dados pode ser vista como um processo de analisar determinados esquemas de relações e chaves primárias para alcançar a minimização da redundância e minimização de anomalias de inserção, exclusão e atualização.

Segundo Navathe em [11] Codd propôs três formas normais que ele chamou de primeira, segunda e terceira forma normal. Além destas, que serão abordadas a seguir, existem também outras formas normais.

Primeira Forma Normal (1FN)

A primeira forma normal (1FN) é considerada parte da definição formal de uma relação no modelo relacional básico. Tem que o domínio de um atributo só deva incluir os valores atômicos, ou seja, simples e indivisíveis, e que o valor de qualquer atributo em uma tupla deve ter um único valor no domínio daquele atributo. A 1FN impede as 'relações dentro das relações' ou as 'relações como valores de atributo dentro de tuplas' [11].

Segunda Forma Normal (2FN)

A segunda forma normal (2FN) é baseada no conceito de dependência funcional total. Uma dependência funcional será uma dependência funcional total se a remoção de qualquer atributo de um conjunto implicar que a dependência não será mais assegurada. Tem-se que um esquema de relação R está na 2FN se todo atributo não primário A em R tem dependência funcional total da chave primária de R. Se o esquema de uma relação não estiver na 2FN, será feita a normalização para a 2FN, por meio da criação de várias relações na 2FN nas quais os atributos não primários só estarão associados à parte da chave primária [11].

Terceira Forma Normal (3FN)

A terceira forma normal (3FN) está baseada no conceito de dependência transitiva. Um esquema de relação R está na 3FN se satisfizer a 2FN e se nenhum atributo não primário de R for transitivamente dependente da chave primária [11].

2.2.2 Transações

Transação é um conjunto de procedimentos, executados em um banco de dados, que o usuário percebe como uma única ação. As transações devem garantir algumas propriedades, essas propriedades desejadas são chamadas de ACID, e elas devem ser garantidas pelo SGBD [11]. As propriedades ACID são as seguintes:

- Atomicidade: Uma transação é uma unidade atômica de processamento, ou ela será executada em sua totalidade ou não será de modo nenhum.
- Consistência: Uma transação terá sua consistência preservada se sua execução completa fizer o banco de dados passar de um estado consistente para outro.
- Isolamento: Uma transação deve ser executada isolada das demais, ou seja, a execução de uma transação não deve sofrer interferência de quaisquer outras transações concorrentes.
- Durabilidade: As mudanças aplicadas ao banco de dados por uma transação efetivada devem persistir no banco de dados, essas mudanças não devem ser perdidas em caso de falha [11].

2.2.3 Linguagem de Consulta(SQL)

Structured Query Language, ou Linguagem de Consulta Estruturada, é a linguagem de pesquisa declarativa padrão para banco de dados relacional. Nos anos 70, os cientistas da computação começaram a desenvolver uma forma de manipular banco de dados, foi a partir dessas pesquisas que surgiu a SQL. Essa linguagem de consulta estruturada funciona provendo aos programadores ou outros usuários de computador maneiras para conseguir extrair informações do banco de dados.

Os comandos possíveis no SQL podem ser agrupados em categorias que são definidas de acordo com as funcionalidades dos comandos[24]:

- DML: Comandos para manipulação dos dados (DML) que servem para selecionar, inserir, atualizar e deletar informações.
- DDL: Comandos para manipulação da estrutura do banco de dados, como criação e exclusão de banco de dados, entre outros.
- DCL: Controle dos usuários por meio da utilização de privilégios.

A linguagem SQL pode ser considerada uma das maiores razões para o sucesso dos bancos de dados relacionais, por ter-se tornado padrão no modelo relacional, os usuários não têm tanta preocupação ao migrar as aplicações originadas por outros tipos de SGBDs.

A SQL foi projetada para ser uma linguagem abrangente, incluindo as declarações em seções para a definição de dados, consultas, visões e especificação de restrições [11].

2.2.4 MySQL

MySQL é um sistema gerenciador de banco de dados do modelo relacional. Este SGBD permite um uso eficiente de armazenamento, procura, ordenação e recuperação de dados, tendo como a SQL sua linguagem de consulta. O MySQL controla o acesso aos dados para garantir que múltiplos usuários possam trabalhar concorrentemente e que somente usuários autorizados possam acessá-los [4].

As principais características do MySQL são:

- Portabilidade: Funciona em diferentes plataformas, ou seja, pode ser utilizado em sistemas Unix, OS X e Windows;
- Tipos de dados: Suporta a maioria dos tipos de dados utilizados, como *INT*, *FLOAT*, *VARCHAR*, *TEXT*, *TIME*, *DATETIME*, entre outros;
- Funções: Possui suporte as diversas funções da linguagem SQL, como *JOIN*, *SELECT*, *WHERE*, *GROUP BY*, entre outros;
- Segurança: Sistema de senha e privilégios que é flexível e seguro, onde a segurança da senha é feita por criptografia;
- Conectividade: Conexão do cliente com o MySQL pode ser feita utilizando diversos protocolos, como *TCP/IP*, no Windows com o uso de *pipe* e no Unix com os arquivos de *socket*;
- Ferramentas: Possui um conjunto de recursos práticos desenvolvidos em cooperação com os usuários.

O MySQL foi o sistema gerenciador de banco de dados do modelo relacional escolhido como objeto de estudo da pesquisa.

2.3 Banco de Dados *NoSQL*

Apesar de ser um termo antigo, o *NoSQL* foi popularizado após o grande crescimento da produção de dados e a consequente necessidade de um novo modelo de banco de dados. Este modelo surgiu para suprir as necessidades que o modelo relacional carecia, como alto desempenho com grande volume de dados, gerenciamento de dados não-estruturados e semi-estruturados e escalabilidade. Grandes empresas estão envolvidas no desenvolvimento do modelo *NoSQL*, tais como *Google*, *Facebook* e *Amazon*, uma vez que estas estavam tendo problemas para lidar com a enorme quantidade de dados que os SGBDs convencionais não conseguiam suportar [21]. Desta forma cada empresa criou seu próprio sistema gerenciador de banco de dados que satisfizesse suas necessidades. O *Facebook* com o *Cassandra*, a *Google* com o *Big Table* [6] e a *Amazon* com o *Dynamo* [6]. Isto acabou gerando uma grande diferença entre os bancos, tanto em sua estrutura interna quanto nas formas de consulta. Bancos de dados relacionais são orientados a transação e se baseiam no conceito de transações com as propriedades ACID. Quando as grandes empresas começaram a ter problemas de escala, perceberam que deveriam abrir mão das propriedades propostas pelo modelo relacional para ganhar escalabilidade. Esses fatos foram estudados posteriormente pelo professor Eric Brewer da Universidade da Califórnia Berkley, resultando na construção do [7] com o "Teorema CAP". Para esse teorema a sigla CAP significa:

- Consistência (*Consistency*): Todos os nós veem o mesmo dado ao mesmo tempo;
- Disponibilidade (*Availability*): Todos os clientes podem ter acesso aos dados quando quiserem. Mesmo se o servidor estiver desligado;
- Tolerante a Partição (*Partition Tolerant*): O sistema continua funcionando mesmo se duas partições perderem a comunicação.

O teorema CAP diz que apenas dois dos três diferentes aspectos podem ser totalmente encontrados em um banco de dados ao mesmo tempo. Não será abordado a fundo o teorema do CAP, porém é importante entender o significado de cada propriedade e observar sua existência em cada banco *NoSQL*. A Figura 2.1 abaixo exemplifica este conceito.

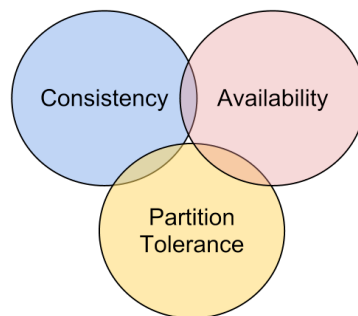


Figura 2.1: Teorema CAP.

Com o crescimento de popularidade das aplicações *web* e o consequente crescimento de transações realizadas em um banco de dados, buscar garantir a consistência dos dados tornou-se um gargalo [22]. Desta forma, para sistemas distribuídos, o protocolo chamado BASE (*Basically Available, Soft-State e Eventual Consistency*) é uma alternativa ao ACID do modelo relacional. Ao contrário do ACID que garante a consistência dos dados para toda transação, o BASE acredita que a consistência do banco de dados está em um estado de flux, ou seja, não é algo estático. A disponibilidade deste protocolo é atingida pois acredita que inconsistências parciais podem acontecer. Pode-se dizer que o *BASE* foca mais na disponibilidade do que na consistência de um banco de dados [14].

2.3.1 Características dos Bancos de Dados *NoSQL*

Existem muitos diferentes bancos de dados *NoSQL*, onde cada um foi criado para solucionar diferentes problemas. Porém esses, em sua maioria, possuem características em comum, que acabam se tornando características dos bancos de dados *NoSQL*, são elas [6]:

- Alta escalabilidade;
- Alta performance;
- Alta disponibilidade;
- Processamento distribuído;

- Ausência de esquema;
- Suporte a replicação de dados;
- Armazenamento de dados estruturados e não estruturados;
- API simples para acesso aos dados;
- Maior flexibilidade às propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade);
- Não suportam linguagem SQL;
- São produtos novos.

Esse novo modelo de armazenamento de dados vem como uma nova alternativa para gerenciar o alto número de dados criados no dia de hoje.

2.3.2 Modelo de Dados *NoSQL*

Existem muitos bancos de dados *NoSQL* diferentes, porém alguns grupos podem possuir semelhanças em suas estruturas internas ou na forma como tratam os dados. Em [18] são 4 as classificações do *NoSQL*: Chave-Valor, baseado em documento, baseado em coluna e baseado em grafo. A seguir, cada um dos modelos é apresentado.

Chave-Valor

Neste modelo, os dados são armazenados como pares chave-valor. Este sistema é semelhante aos dicionários, onde os dados são endereçados por uma chave. Os valores são isolados e independentes, e o relacionamento é tratado pela lógica da aplicação. O banco de dados chave-valor é livre de esquema, e é aceitável existir valores diferentes de dados no momento da leitura. A aplicação é a responsável por resolver o problema de incompatibilidade, caso exista. Este tipo de banco de dados é útil para aplicações em que o processamento das transações é baseado em chaves e para aplicações que realizam constantes leituras nos dados [14]. O banco de dados chave-valor é uma solução adequada para aplicações simples que funcionam com um único tipo de objeto, e esses objetos são baseados em um único atributo [21]. Exemplos de banco de dados chave-valor são: *Toquio Tyrant*, *Berkeley DB*, *Voldemort*, *Dynamo*, *Riak* e *Redis*.

Baseado em colunas

Os bancos de dados baseados em coluna definem a estrutura dos valores como um conjunto pré-definido de colunas. A família de coluna pode ser considerada como um esquema de base de dados, onde as estruturas super colunas e família de colunas determinam o esquema do banco de dados [14]. As principais definições são coluna, super colunas e família de colunas:

- A coluna é uma informação atômica do banco de dados. Ela é expressa por nome e valor;

- As super colunas associam colunas para conjuntamente retornar dados a partir do disco ou fazer associação semântica. Esse modelo é útil para tipos de dados complexos, tais como endereços que são compostos pela conjunção de caracteres de texto e número;
- Uma família de colunas une colunas e super colunas dentro de uma estrutura complexa. O conceito desse modelo lembra uma tabela do modelo relacional.

O banco de dados família de colunas não é um modelo restritivo: novas colunas ou super colunas podem ser facilmente adicionadas às bases de dados nos ambientes de produção. Cada linha em um banco de dados família de colunas não necessita possuir o mesmo grau, ou seja, pode ter um número variável de colunas e super colunas. Devido a esta característica, esse modelo de banco de dados apoia dados espaçados. As colunas de uma tabela estão divididas sobre os nós usando o conceito de grupos de colunas. Parece complexo, mas grupos de colunas são uma maneira simples dos usuários indicarem quais colunas devem ser armazenadas juntas [21].

Este modelo é adequado para aplicações que tratam com grandes volumes de dados, de modo que o modelo de dados pode ser eficientemente particionado [19]. De acordo com Catell em [21] para os casos onde se deseja otimizar a leitura de dados estruturados, o modelo família de colunas é mais interessante, pois mantém os dados de forma contígua por coluna. [15] Kaur e Rani (2013) consideram esses bancos de dados apropriados para propósitos analíticos porque eles podem tratar com colunas específicas.

Exemplos de banco de dados família de colunas são *BigTable*, *HBase*, *PNUTS* e *Cassandra*.

Baseado em documento

Nesse modelo, os documentos são coleções de atributos e valores, onde um atributo pode ser multivalorado. Cada documento contém um identificador, que é único dentro da coleção. O armazenamento de dados é do tipo ISDN (*Integrated Services Digital Network*) e tem uma vantagem adicional no suporte aos tipos de dados, o que torna os bancos de dados orientado a documentos mais amigáveis para se utilizar.

Em geral, o banco de dados orientado a documentos não possui esquema: o documento não precisa ter uma estrutura comum. O modelo orientado a documentos costuma armazenar os valores em uma estrutura como JSON (*JavaScript Object Notation*) ou XML (*Extensible Markup Language*). O formato JSON suporta os tipos de dados *list*, *map*, *date*, *boolean* e também números com diferentes precisões [14]. O formato XML também suporta vários tipos de dados.

Catell em [21] aponta que ao contrário do modelo chave-valor, o modelo orientado a documentos geralmente suporta índices secundários e diversos tipos de documentos (objetos) por banco de dados, além de documentos aninhados ou listas.

Hecht e Jablonski em [19] afirmam que o modelo orientado a documentos é de fácil manutenção e são, portanto, desejáveis para aplicações *web* que precisam executar consultas dinâmicas, tais como aplicações de análise em tempo real, e *blogs*. Um exemplo de consulta dinâmica é quando se tem diferentes tipos de objetos, e é necessário pesquisar objetos com base em múltiplos campos.

Exemplos de banco de dados orientado a documentos são: *MongoDB*, *CouchDB* e *Riak*.

Baseado em grafo

Um banco de dados baseado em grafos utiliza o modelo de grafos para representar o esquema. Esse modelo funciona com três abstrações: nós, as relações entre nós e o par chave-valor anexado aos nós e relacionamentos [5](Tauro and Shreeharsha, 2012).

Esse modelo está diretamente relacionado a teoria de grafos e suporta a utilização de restrições de integridade de dados, tais como restrições de identidade e integridade referencial, garantindo assim as relações entre elementos de forma consistente. Devido a sólida teoria de grafos existem linguagens de consulta disponíveis que podem ser utilizadas, tais como *Cypher*, *Gremlin* e *SPARQL* [15].

Exemplos de bases de dados baseada em grafos são: *Neo4j*, *GraphDB* e *InfoGrid*.

2.4 Projeto e Administração de Banco de Dados

A administração de banco de dados é a função de gerenciar e manter os SGBDs, a fim de garantir que o banco de dados esteja sempre disponível, atualizado e seguro. O responsável por todas essas tarefas é o Administrador de Banco de Dados (DBA). O DBA é a autoridade central para gerenciar o banco de dados, ele possui um entendimento completo da estrutura deste assim como dos dados e seus relacionamentos ali presentes. Desta forma é importante que ele conheça o projeto, a fim de escolher a melhor forma de se realizar determinadas funções. Outra função importante é a do projetista de banco de dados, que trabalha com a modelagem deste, buscando construir o melhor design de banco de dados para uma aplicação.

O uso de linhas de comando, *softwares* e interfaces de administração do banco de dados faz parte da rotina administrativa de qualquer banco de dados. Uma linguagem de consulta bem estabelecida é considerada uma ferramenta de apoio às tarefas de administração de bancos de dados. A linguagem padrão suportada pela grande maioria das bases de dados tradicionais é a linguagem SQL, no qual é possível submeter comandos diretamente ao banco de dados.

Para assumir estas responsabilidades, é necessário muito conhecimento e consequentemente, muito estudo. São cargos críticos e de extrema importância. Dessa forma devem saber trabalhar sobre pressão assim como conseguir tomar decisões corretas mesmo em momentos críticos. Outras habilidades necessárias são:

- Boa comunicação;
- Conhecimento da teoria de Banco de Dados;
- Conhecimento do *design* do Banco de Dados;
- Conhecimento do SGBD usado;
- Conhecimento da linguagem relacionada ao banco de dados;
- Conhecimento de arquitetura distribuídas;
- Conhecimento geral de Sistemas Operacionais;
- Estratégia de *backup* e recuperação de dados;
- Conhecimento de linguagem de programação.

2.4.1 Responsabilidades e Tarefas de um DBA

Segundo Mullins [20] os dados em uma organização devem ser úteis, usáveis, disponíveis e corretos. Sendo assim, isso exige certas responsabilidades para o administrador de banco de dados. Segundo Ramakrishnan [23] as responsabilidades de um DBA podem ser divididas em 4 áreas:

- **Segurança e Autorização:** O DBA tem como responsabilidade garantir que os dados apenas sejam utilizados (leitura e escrita) por quem tem permissão para isso. Por vezes um usuário pode ter acesso aos dados de uma tabela, mas não tem permissão para localizar dados de outra tabela. O DBA deve garantir que isso aconteça corretamente;
- **Disponibilidade de Dados e Recuperação de Falhas:** O DBA deve garantir que em caso de falhas os usuários tenham acesso ao máximo possível de dados não corrompidos, e deve, o mais rápido possível, restaurar o banco de dados para um estado consistente. É também responsável por implementar os procedimentos para realizar *backups* com frequência e manter os *logs* das atividades do sistema a fim de facilitar a recuperação em caso de falha;
- **Sintonização de Banco de Dados:** O DBA deve garantir que o banco de dados acompanhe a evolução do sistema realizando *upgrades*, principalmente nos modelos conceitual e físico, buscando melhorar o desempenho conforme os requisitos são alterados.

Sabendo das responsabilidades, são definidas as tarefas de administração que um DBA deve realizar. Estas envolvem atividades, tais como as descritas em [20] e [25], algumas delas são explicadas a seguir:

- **Controle de acesso:** Esta tarefa busca garantir a segurança do banco de dados. Um SGBD é utilizado por diversos usuários que o utilizam para diferentes finalidades, é tarefa do DBA ter controle de quais usuários estão acessando o SGBD, sendo de sua responsabilidade também a criação e deleção de usuários;
- **Concessão e revogação de privilégios:** Também relacionada com a segurança do SGBD, esta tarefa busca garantir que os usuários possam interagir apenas com tabelas e bancos de dados necessários para o seu trabalho. Sendo assim é possível por meio de privilégios, limitar ou permitir que um usuário realize uma atividade dentro de um banco de dados. Geralmente o DBA tem controle tanto de quais ações um usuário pode realizar no SGBD quanto o local que estas podem ser feitas;
- **Auditoria do banco de dados:** Quanto maior o número de usuários utilizando um SGBD, maior a possibilidade que erros aconteçam, desta forma é importante que o DBA tenha o controle de quais usuários estão realizando o que no SGBD, mantendo um histórico disso, para caso alguma falha aconteça seja possível analisar a causa do problema, facilitando sua recuperação;
- **Migração de estrutura e dados entre bancos de dados:** Por vezes é necessário mudar um SGBD de máquina, ou até mesmo mudar de SGBD, e é tarefa do DBA realizar a migração das estruturas e dos dados entre os bancos de dados, garantindo a consistência dos dados;

- Monitoramento do uso de recursos de *software* e *hardware*: O banco de dados é utilizado continuamente dentro de uma empresa, sendo assim é importante manter um controle constante do uso de *hardware* e *software* por esse. Isso busca evitar gastos desnecessários com *hardware*, assim como evita possíveis falhas;
- Melhoria de desempenho: Um banco de dados deve estar em constante desenvolvimento a fim de acompanhar a evolução de uma aplicação, mudanças acontecem constantemente em um banco de dados e é importante manter um controle do desempenho do mesmo. Por vezes uma aplicação se torna lenta por conta de um gargalo existente no banco de dados. O DBA deve garantir o melhor desempenho possível do banco de dados para melhor favorecer a aplicação. O DBA deve também garantir que o banco de dados não se torne lento, ou até mesmo caia, caso o número de acessos a ele aumente, devendo então garantir uma constante disponibilidade, uma forma de se garantir alta disponibilidade é por meio da replicação de dados;
- Estratégia de *backup* e recuperação de dados: Caso problemas aconteçam com o banco de dados é importante que se possua uma versão atualizada deste para se utilizar, sendo assim o DBA deve ter uma estratégia para realizar *backups* constantes dos bancos de dados e garanta que estes possam ser recuperados rapidamente em caso de falhas.

2.4.2 Responsabilidades e Tarefas de um Projetista

O Projetista de banco de dados deve entender quais dados devem ser armazenados no SGBD e como eles serão utilizados. Sabendo disso, ele pode projetar o esquema conceitual, decidindo quais relações armazenar, e o esquema físico onde ele decide quais dados armazenar. Este busca diminuir a redundância e a duplicidade, a fim de garantir um banco de dados com armazenamento consistente e consultas eficientes.

A principal tarefa do projetista é a modelagem do banco de dados (Conceitual, lógico e físico). Esta atividade é realizada por um processo que constrói três modelos, conceitual, lógico e físico. Busca entender os dados utilizados por uma aplicação, e como estes se relacionam para obter um banco de dados que mais se adeque a mesma. O modelo conceitual é o mais alto nível e por vezes independente de tecnologia, ele busca entender as entidades da aplicação, seus atributos e como estas se relacionam umas com as outras. A abordagem entidade-relacionamento (ER) é a técnica para modelagem de dados mais difundida e utilizada para o modelo conceitual, e representada graficamente através do diagrama entidade-relacionamento (DER). A abordagem ER foi criada em 1976 por Peter Chen e pode ser considerada como um padrão de fato para a modelagem conceitual. Existem alguns conceitos centrais da abordagem ER que são entidade, relacionamento, atributo e cardinalidade [12]. Entidade é um conjunto de objetos da realidade modelada sobre os quais deseja-se manter informações no banco de dados. Relacionamento é um conjunto de associações entre entidades. Atributo é o dado que é associado a cada ocorrência de uma entidade ou de um relacionamento. Por fim cardinalidade é o número de ocorrências de entidade associadas a uma ocorrência da entidade em questão através do relacionamento [12]. O modelo lógico tem como objetivo, utilizar os dados criados no modelo relacional para construir tabelas, nesta etapa são definidas as chaves para cada tabela, e novas tabelas são criadas para representar certos relacionamentos. Por fim o mo-

delo físico adiciona tipos aos atributos de cada tabela do modelo lógico e busca melhorias neste a fim de se obter mais eficiência.

2.4.3 Tipos de DBA

Um banco de dados pode possuir diversos tamanhos e diferentes complexidades, onde para bancos de dados muito grandes e complexos é inviável que apenas um DBA cuide de sua totalidade, sendo assim é possível dividir as funções do DBA em uma equipe mais especializada que irá trabalhar em conjunto para conseguir obter um maior êxito na administração do banco de dados. Então as tarefas e responsabilidades do DBA podem ser divididas em alguns tipos. As empresas responsáveis pelos SGBDs estão cada vez mais oferecendo certificações específicas nas áreas de DBA, para capacitar este profissional, tornando-o mais especializado em determinadas tarefas dos administradores de banco de dados. Alguns dos tipos de DBA existentes são:

- **Administrador do Sistema:** Esse DBA é um tipo de administrador geral do sistema, sendo assim esse profissional tem que ter um pouco de conhecimento nas diferentes áreas do banco de dados, conhecendo a fundo tudo do sistema que ele está participando. Em projetos pequenos é comum ter esse tipo de DBA, que faz desde a modelagem, até a implementação do banco de dados, realiza o *backup* e garante a segurança do sistema. Existe certificações para esse tipo de DBA, que vão desde etapas básicas e mais simples, das funções desse DBA, até as mais difíceis e complexas, o que demonstra que esse profissional precisa conhecer todas as áreas e o sistema que ele está inserido;
- **Desenvolvedor:** Esse tipo de DBA escreve o código, ou seja, é escreve os *scripts*, grava os procedimentos, funções, ou qualquer tipo de código que pode ser utilizado para acesso ao banco de dados. Em projetos maiores é comum esse DBA trabalhar junto com os outros tipos de DBA, principalmente o *Designer*;
- **Especialista em Alta Disponibilidade:** Esse tipo de DBA é mais recente, foi criado por algumas organizações que precisavam de alguém especializado em alta disponibilidade. Sendo assim esse profissional tem como objetivo determinar quais os métodos de alta disponibilidade são os melhores para a aplicação e a partir disso realizar sua implementação e sua manutenção;
- **Especialista em Segurança:** Esse tipo de DBA é responsável pela segurança do banco de dados, com a enorme quantidade de dados, e dados importantes, um profissional especializado nessa área tornou-se mais importante, sendo assim ele é responsável por prover o controle de acesso, quem poderá ler e escrever nas tabelas, quem tem acesso aos dados. Em projetos maiores é mais comum esse tipo de DBA. Devido a crescente necessidade de segurança, está se tornando comum o treinamento e a certificação de DBA nessa área de segurança.

Capítulo 3

NoSQL Cassandra

Este capítulo apresenta o banco de dados *NoSQL* Cassandra, objeto de estudo nessa pesquisa. Nas seções 3.1 e 3.2 é realizada uma breve descrição deste banco de dados, descrevendo suas principais características, abordando sua arquitetura, mostrando como esta funciona além de apresentar alguns conceitos importantes. Na seção 3.3 é abordada o modelo de dados do Cassandra, tratando sobre o *cluster*, *keyspace*, coluna de família, linhas e colunas assim como as operações de escrita e leitura. Por fim na seção 3.4 é estudada a linguagem CQL (*Cassandra Query Language*) utilizada por esse sistema.

3.1 O Cassandra

O *Apache* Cassandra é um banco de dados NoSQL *open source* escalável. É utilizado para gerenciar dados estruturados, semiestruturados e não estruturados sobre múltiplos centros de dados e na nuvem. O Cassandra oferece disponibilidade contínua, escalabilidade linear e um operacional simples por meio de muitos servidores sem um único ponto de falha, com uma arquitetura *masterless* em formato de anel, onde não existe um nó principal. Isto funciona juntamente com um modelo de dados dinâmico, criado para obter máxima flexibilidade e rápidas respostas [1]. Algumas características do Cassandra são:

- Distribuição de dados automática: Distribuição automática dos dados sobre todos os nós participantes do anel ou *cluster*. Não há necessidade do administrador ou programador fazer ou programar nada para distribuir os dados sobre o *cluster* uma vez que o dado é transparentemente particionado sobre todos os nós;
- Replicação customizável: Capacidade de armazenar cópias redundantes de um dado sobre nós de um mesmo anel. Isso significa que, se qualquer nó de um *cluster* falhar, uma ou mais cópias dos dados deste nó estarão disponíveis em outra máquina deste mesmo *cluster*. A replicação pode ser configurada para funcionar sobre um *data center*, vários *data centers* ou múltiplos *cloud zones* disponíveis;
- Escalabilidade linear: A capacidade do banco de dados pode ser facilmente aumentada simplesmente adicionando novos nós *online*. Por exemplo, se 2 nós são capazes de suportar 100.000 transações por segundo, 4 nós suportarão 200.000 transações por segundo e 8 nós 400.000 transações por segundo, assim como mostrado na Figura 3.1.



Figura 3.1: Cassandra: Escalabilidade Linear [1].

3.2 Arquitetura

Em bancos de dados distribuídos tradicionais existe uma relação de escravo-mestre entre os nós. O mestre é responsável por distribuir e gerenciar os dados e os escravos sincronizam seus dados com os do mestre. Tudo que é escrito no banco passa pelo mestre, que é um ponto único de falha. A arquitetura mestre-escravo tem um lado negativo, que é no caso de algum problema acontecer no nó mestre [1].

O Cassandra, pelo contrário, entende que falhas acontecem e por isso utiliza o modelo distribuído *peer-to-peer*, aonde os dados são divididos entre todos os nós no *cluster*, que possuem estruturas idênticas. Todos esses nós em um *cluster* podem aceitar tanto leituras quanto escritas e, frequentemente trocam informações sobre seu estado e de outros nós do *cluster* usando o protocolo de comunicação *peer-to-peer gossip*, possibilitando que estejam atualizados sem a necessidade de um coordenador mestre.

O Cassandra é um banco de dados de linhas particionadas, onde as linhas são organizadas em tabelas com uma chave primária obrigatória, a *partition key*. Esta arquitetura permite que qualquer usuário autorizado se conecte com qualquer nó, em qualquer *data center*, e acesse os dados usando a linguagem *CQL*.

Requisições de escrita e leitura de clientes podem ser enviadas à qualquer nó no *cluster*. Quando um cliente se conecta com um nó por meio de uma requisição, esse nó serve de coordenador para essa operação e age como um *proxy* entre a aplicação do cliente e o nó que possui o dado.

Essa arquitetura é apresentada na Figura 3.2, onde *gossip* é a forma em que os diferentes nós se comunicam, e A, B, C e D são os nós em si. Isso permite que os dados sejam replicados em partições (*Clusters*), que podem estar distribuídos em diferentes *data centers*.

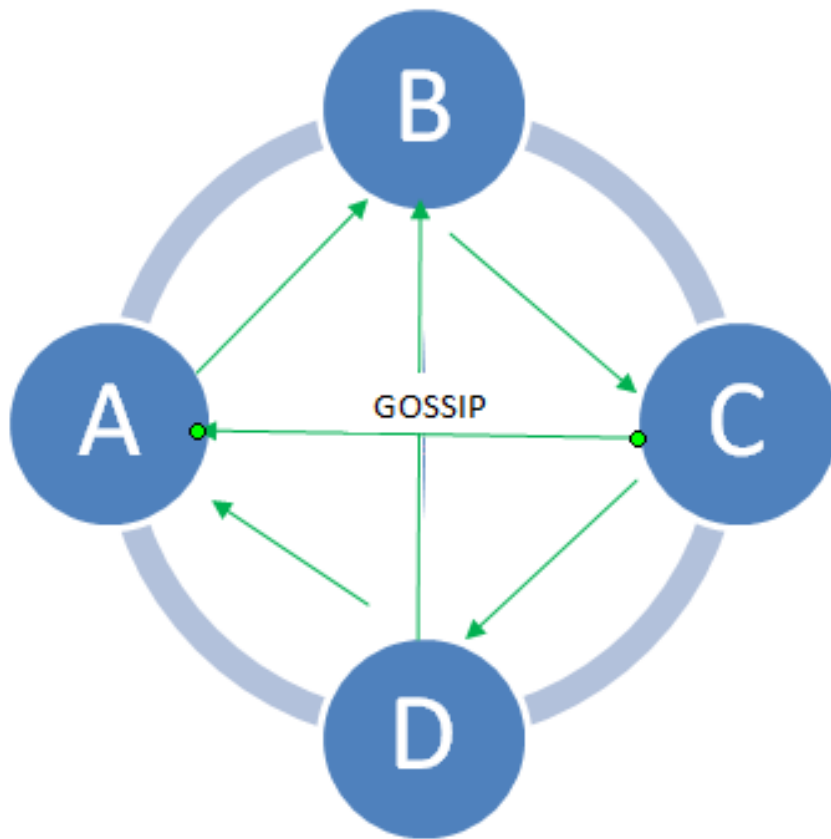


Figura 3.2: Arquitetura Cassandra.

Existem muitas construções teóricas e estruturas que suportam a arquitetura do Cassandra. Primeiro são apresentadas algumas estruturas básicas utilizadas pelo Cassandra, para então passar por alguns princípios importantes.

3.2.1 Estruturas Básicas Principais

A estrutura do Cassandra é baseado em [1]:

- **Nó:** Aonde o dado é armazenado. É a infraestrutura básica do Cassandra;
- **Data center:** Uma coleção de nós relacionados. É dividido em *data center* físico e virtual. A replicação é feita pelos *data centers*;
- **Cluster:** Contém um ou mais centro de dados;
- **Commit Log:** Todo dado é escrito primeiramente no *commit log* para garantir durabilidade. Depois que todo dado foi escrito no *SSTable*, ele pode ser arquivado, deletado ou reciclado;
- **SSTable:** Um *SSTable* (*Sorted String Table*) é um arquivo de dados imutável aonde o Cassandra escreve dados periodicamente. Ele é armazenado em disco e mantido para cada tabela do Cassandra;

- *CQL Table*: Coleção de colunas ordenadas indexadas pela linha da tabela. A tabela consiste de colunas e possui uma chave primária.

3.2.2 Princípios importantes

O Cassandra foi projetado seguindo alguns princípios computacionais e cada um destes é descrito a seguir.

Gossip

Protocolo *peer-to-peer* de comunicação para descobrir e compartilhar a localização e o estado sobre os nós que participam de um *cluster* do Cassandra. Quando um nó é inicializado seu arquivo de configuração determinará qual o nome do *cluster* que o nó pertence. Informará também com quais nós *seeds* (nós que mantêm informação sobre todos os outros nós do *cluster*) devem ser feito contato, e determinará ainda outros parâmetros, como porta e faixas. As informações desse protocolo são guardadas localmente por cada nó para usar imediatamente quando um nó reinicia. Periodicamente cada nó troca informação sobre o seu estado e dos outros nós que eles "conversaram". O processo de *gossip* acontece todo segundo e as mensagens podem ser trocadas por até três nós no *cluster* isso faz com que rapidamente todos nós saibam sobre todos os nós do *cluster*. Uma mensagem *gossip* tem uma versão associada à ela, assim quando um versão mais nova chega essa pode sobrescrever uma mais antiga [1].

Descoberta e recuperação de falhas

Detecção de falhas: Método para localizar localmente, por meio do estado *gossip* e histórico, se um nó do sistema caiu ou voltou a funcionar. Cassandra usa essa informação para evitar enviar requisição de cliente para nós que não podem ser alcançados. O processo de *gossip* rastreia o estado de outros nós conectados tanto diretamente quanto indiretamente. Cassandra utiliza um mecanismo de detecção de desempenho para marcar os nós que não devem ser utilizados, esse mecanismo leva em consideração a performance da rede, *workload*, e condições históricas. É possível configurar a sensibilidade do detector de falhas, sendo assim caso esteja usando um servidor mais congestionado, como o da *Amazon*, é possível diminuir essa sensibilidade uma vez que a falha pode ocorrer por conta do congestionamento e não por falha do nó.

Os nós podem falhar por diversos motivos como problemas no *hardware* ou interrupção na rede. As interrupções na rede geralmente são breves, sendo assim, ela raramente significa que o nó deva sair permanentemente do *cluster*. Caso uma interrupção na rede aconteça o nó não é automaticamente retirado do anel, os outros nós tentarão reestabelecer a conexão periodicamente para ver se ele voltou a funcionar. Para remover definitivamente um nó do *cluster* o administrador deve explicitamente fazê-lo.

Quando o nó retorna de uma interrupção ele pode ter perdido alguma inserção das réplicas que são mantidas. Para recuperar dados perdidos existe mecanismos de reparo como o *hinted handoff* ou reparo manual utilizando o *nodetool repair*. O tamanho da interrupção vai definir qual o melhor mecanismo a ser utilizado para garantir a consistência dos dados [1].

Hinted handoff: Em uma transação, caso o nó que deveria receber um dado esteja fora, *hints* são escritos no nó coordenador, contendo os dados envolvidos e o nó indisponível. Consequentemente quando este voltar, os dados poderão ser escritos [1].

Node tool Repair: Também utilizado para manter a consistência dos dados, este processo monta uma árvore binária com os dados do nós (*Merkle Trees*) contendo um valor *hash* das linhas que serão comparadas afim de garantir a consistência dos dados. Este processo pode acontecer por nó, ou por faixa de *token* [1].

Distribuição

No Cassandra, a distribuição e replicação de dados são realizados juntos. Os dados são organizados por tabelas (*hashing table*) e identificados por chave primária (*partition key*), que determinam em qual nó o dado está armazenado. Réplicas são cópias de linhas distribuídas nos nós. Quando o dado é primeiramente escrito ele é também identificado como réplica. Fatores que influenciam a replicação são [1]:

- Nó virtual: Atribui a propriedade dos dados para as máquinas físicas;
- Partição: Determina como o dado será particionado ao longo do *cluster*;
- Estratégia de replicação: Determina como a replicação será feita para cada linha de dados;
- *Snitch*: Define as informações de topologia que a estratégia de replicação usa para replicar os dados.

Para facilitar a distribuição dos dados, a fim de garantir escalabilidade e fácil acesso aos dados, o Cassandra utiliza duas estruturas, o *hashing* consistente e os nós virtuais:

Hashing consistente: *Hashing* consistente permite a distribuição do dado no *cluster* e minimiza a reorganização quando um nó é adicionado ou removido. Cada linha da tabela no modelo de dados, tem uma chave de partição que a identifica, esta é responsável pela distribuição dos dados no *cluster*. O Cassandra atribui um valor *hash* para cada uma dessas chaves e cada nó é então responsável por uma faixa de *hashs*, e será responsável pelo dado dependendo dessa faixa. A Figura 3.3 demonstra como funciona o *hashing* consistente nos nós, como pode ser observado tem 4 nós no anel onde cada nó é responsável por uma faixa de *hashs* [1].

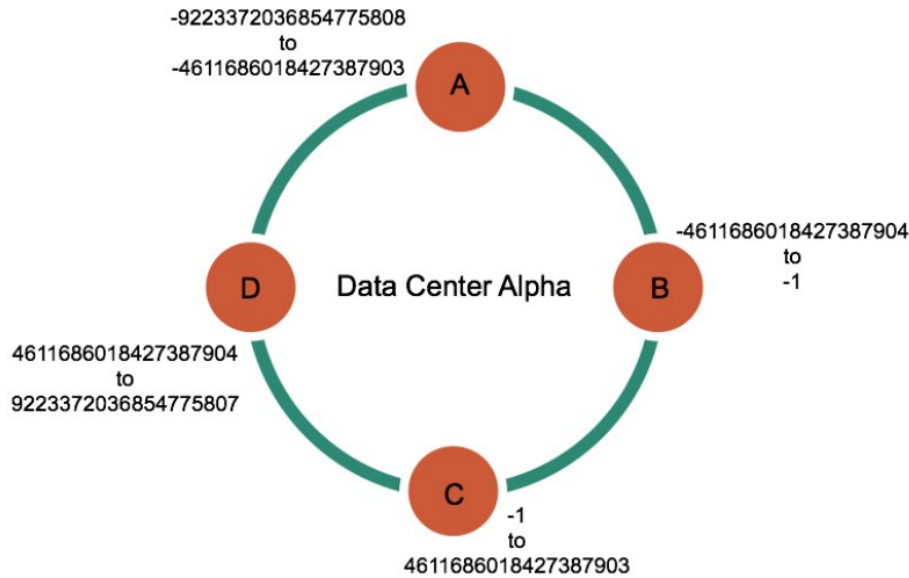


Figura 3.3: Exemplo hashing consistente.

Token: *Token* é o início de uma faixa de *hashs*. O Cassandra decide aonde colocar cada dado atribuindo os *tokens* aos nós. Até a versão 1.2 do Cassandra cada nó possuía um *token*, porém isso dificultava para adicionar e deletar nós pois gerava a necessidade de recalculer os *tokens*. Criou-se então os nós virtuais para facilitar esse processo [1].

Nós virtuais: Nós virtuais são utilizados para distribuir os dados nos nós de um *cluster*, é o conceito de cada nó aceitar mais de um *token*. Com nó virtual, também chamado de *Vnode*, um nó possui um grande número de pequenas partições distribuídas por todo *cluster*. A figura abaixo mostra a diferença entre um anel sem nó virtual e outro com nó virtual. A primeira parte mostra um *cluster* que não utiliza *Vnode* nesse paradigma para cada nó é atribuído apenas um *token* que representa a localização do nó no anel onde cada nó guarda o dado mapeando a chave de partição do dado com um valor do *token* dentro da faixa do nó. Cada nó também contém uma cópia dos outros *tokens* do *cluster* dependendo do fator de replicação. A segunda parte mostra um anel que utiliza *Vnodes*, sendo assim, dentro de um *cluster* os *Vnodes* são escolhidos aleatoriamente e não são contíguos. A localização de uma linha é determinada pelo *hash* da chave de partição associada à um *token*, que é uma faixa de partições pertencentes a cada nó [1].

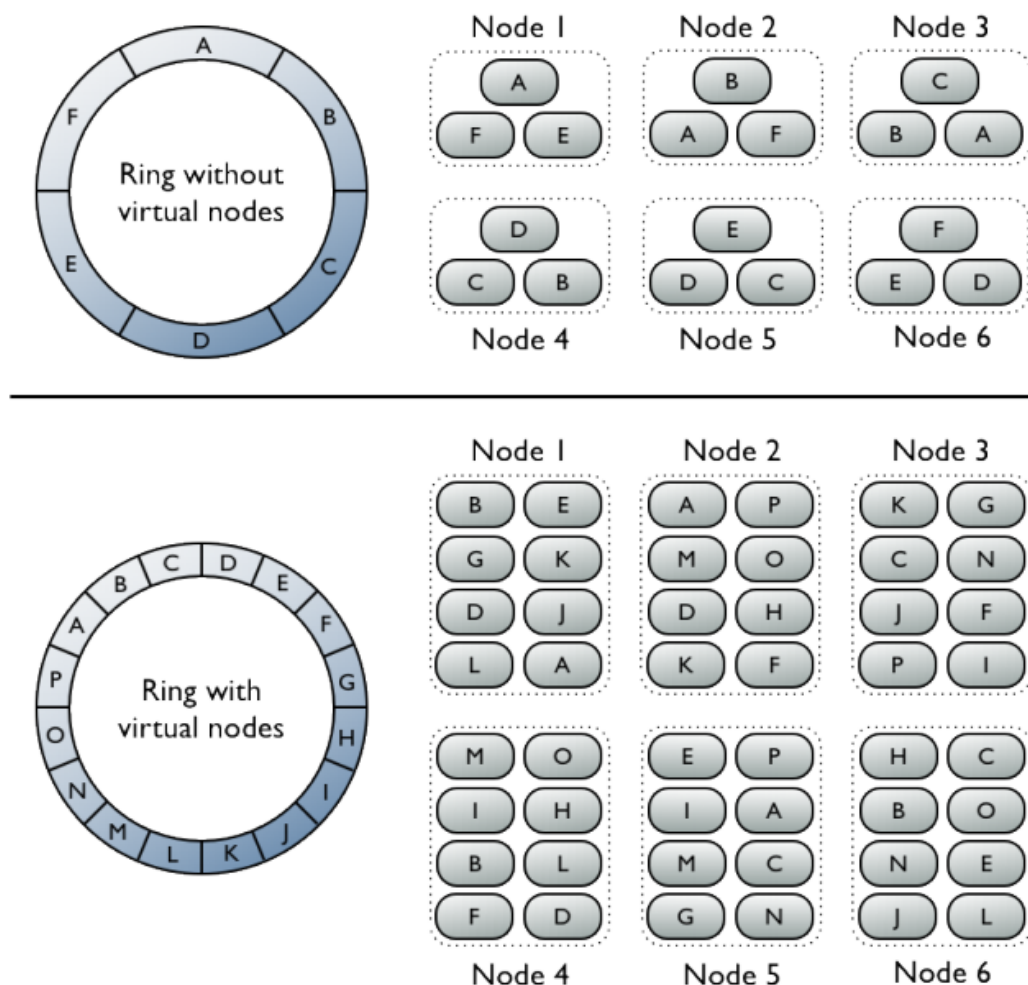


Figura 3.4: Arquitetura Vnode vs token único.

Replicação dos Dados

Cassandra guarda réplicas dos dados em múltiplos nós para garantir disponibilidade e tolerância a falhas. A estratégia de replicação (*replication strategy*) determina o nó aonde as réplicas serão alocadas. A quantidade total de réplicas no *cluster* é definida pelo fator de replicação (*replication factor*), onde um fator de replicação de 2 significa que serão feitas 2 cópias de cada linha onde cada cópia é colocada em um nó diferente. Todas as réplicas são igualmente importantes pois não existe uma réplica primária ou *master*. Como uma regra geral o fator de replicação não deve exceder o número de nós do *cluster* e é interessante que seja maior do que um. O Cassandra oferece duas estratégias de replicação [1]:

- *SimpleStrategy*: Utiliza apenas um *data center*. Aloca a primeira réplica no nó determinado pelo particionador. As próximas réplicas são alocadas nos nós vizinhos do anel no sentido horário;
- *NetworkTopologyStrategy*: Mais recomendada pois é mais simples de ampliar para múltiplos *data centers* quando futuras expansões forem necessárias.

Particionador: Um particionador determina como os dados, incluindo réplicas, serão distribuídos entre os nós de um *cluster*. Um particionador é uma função *hash* que calcula o *token* de uma chave de partição, tipicamente por meio de *hashing*. Cada linha de dado é identificada por uma única chave de partição e distribuída no *cluster* pelo valor do *token*. Existem três tipos de particionadores oferecidos pelo Cassandra [1]:

- *Murmur3Partitioner*: Distribui uniformemente os dados no *cluster* utilizando a função *hash MurmurHash*;
- *RandomPartitioner*: Distribui uniformemente os dados no *cluster* utilizando a função *hash MD5*;
- *ByteOrderedPartitioner*: Mantém a distribuição ordenada dos dados pelos *bytes* das chaves.

A configuração do particionamento pode ser realizada pelo arquivo *cassandra.yaml* e por *default* é utilizado o *Murmur3Partitioner* [1].

Snitch: O *snitch* determina a partir de quais *data centers* e *racks* os dados serão lidos e escritos. O *snitch* informa sobre a topologia da rede, de modo que as solicitações são encaminhadas de forma eficiente e permite distribuir réplicas por agrupamento de máquinas em *data centers* e *racks*. A propriedade *snitching* dinâmico monitora o desempenho de leituras das réplicas e escolhe a melhor réplica com base no histórico [1]. Abaixo são listados os tipos de *snitches* disponíveis:

- *SimpleSnitch*;
- *RackInferringSnitch*;
- *PropertyFileSnitch*;
- *GossipingPropertyFileSnitch*;
- *EC2Snitch*;
- *EC2MultiRegionSnitch*.

Bloom Filter: *Bloom filter* é um espaço de eficiência probabilística de estrutura de dados usado para melhorar performance. *Bloom filter* analisa se um elemento é membro de um grupo usando um rápido e não determinístico algoritmo. Esses filtros são parte da memória e agem fazendo pesquisas rápidas, é utilizado para determinar se uma linha está em uma *SSTable* [1].

Consistência dos dados

Operações de reparo contínuo garantem que todos as réplicas de uma linha eventualmente estarão consistentes. Esses reparos servem para diminuir a variabilidade dos dados replicados, mas um alto tráfego de dados sobre um grande sistema distribuído pode gerar inconsistências. De acordo com o teorema CAP apresentado anteriormente, o Cassandra se encaixa como um sistema AP pois é altamente disponível e tolerante à partição, porém dependendo da configuração, o Cassandra pode vir a se tornar um sistema CP. Sendo

assim, é possível variar o nível de consistência dos dados, tendo consequência sobre a disponibilidade dos mesmos.

Configuração de consistência: No Cassandra, cada dado é replicado em N locais onde N é definido como o fator de replicação configurado para cada instância. A replicação ajuda o Cassandra a ser mais disponível e durável. O Cassandra também é bem consistente e diferentes níveis de consistência podem ser criados para cada transação. Podem ser zero, um, qualquer, *quorum* ou todos. Se o QUORUM é definido como consistência de escrita, a escrita é recebida por pelo menos a maioria das réplicas. Cassandra pode manter suas réplicas consistentes uma com a outra em cenários onde não há falhas no servidor [1].

Read repair: Para cada consulta realizada um nó busca todas as réplicas para o dado, apenas o primeiro fornece o dado completo e o resto em *hash*, caso o número configurado forem idênticos, este é retornado. Caso exista inconsistência, é requisitado o dado completo de todas as réplicas, e a mais recente é atualizada em todos os casos.

3.3 Modelo de Dados

O modelo de dados no Cassandra se preocupa em como o dado será consultado pela aplicação e busca preparar seu modelo de dados para atender as consultas com apenas uma tabela [8]. Uma das consequências disso é que um dado estará em mais de uma tabela, com isso serão armazenados muito mais dados em disco do que necessário quando comparado com o modelo relacional, mas isso é recompensado uma vez que as consultas serão facilitadas. O dado no banco de dados Cassandra é formado por uma coleção de colunas, gerando uma linha, esta é armazenada em tabelas distribuídas e multidimensionais, *indexadas* por uma chave *hash*, que estão agrupadas em um *keyspace* que por fim está presente em um *cluster*. Cada uma dessa estrutura será aprofundada abaixo.

3.3.1 Cluster

O banco de dados Cassandra possui um *design* que o permite ser distribuído sobre diversas máquinas funcionando juntas de forma que pareça apenas uma para o usuário final. O Cassandra organiza os nós (servidores) no *Cluster* em formato de anel, como explicado na sessão anterior. Um nó possui réplicas de diferentes faixas de dados, caso um nó fique fora do ar, uma réplica em outro nó pode responder as consultas. Os nós se comunicam utilizando o protocolo *peer-to-peer gossip* [13].

3.3.2 Keyspace

O *Keyspace* funciona como o banco de dados quando comparado ao modelo relacional, e é a mais externa das estruturas para os dados no Cassandra. Apesar de ser possível criar múltiplos *keyspaces*, é recomendado que se utilize apenas um, a criação de um número muito grande de *keyspaces* pode causar problemas [13]. Os atributos básicos que podem ser configurados para o *keyspace* são:

- *Replication Factor*: A quantidade de nós que receberão cópia de cada linha de dado;
- *Replica placement strategy*: Se refere a como as réplicas serão posicionadas no anel. Existem diferentes estratégias para definir e para determinar quais nós receberão

cópias de quais dados. Este, juntamente com o fator de replicação são definidos na criação do *keyspace*;

- *Column Family*: Assim como o banco de dados contém diversas tabelas no modelo relacional, o *keyspace* contém uma lista de uma ou mais *column families*.

3.3.3 Família de Colunas

A família de colunas, também chamada de tabela, pode ser vista como um conjunto de partições que possuem linhas com estruturas similares [13]. Cada partição em uma tabela possui uma única chave de partição e cada linha em uma partição pode opcionalmente ter uma chave do *cluster* (*clustering key*). A combinação da *partition key* com a *clustering key* identifica unicamente uma linha em uma tabela e é chamada chave primária, sendo a *partition key* obrigatória, enquanto a *clustering key* é opcional.

Dessa forma uma tabela sem *clustering key* podem ter apenas uma linha por partição, uma vez que a chave primária é definida pela chave de partição. Por outro lado, uma tabela com *clustering key* pode possuir múltiplas linhas em uma partição onde as diferentes linhas de uma mesma partição terão diferentes *clustering key*, e essas linhas serão ordenadas de acordo com essa chave, por *default* em ordem crescente [8].

A criação da tabela em CQL define um conjunto de colunas e uma chave primária, para cada coluna é assinalado um tipo, que pode ser primitivo, como *int* ou *text*, ou complexo, como conjunto, lista ou mapa. Uma coluna pode ser definida como estática para definir um valor que será compartilhado por todas as linhas de uma mesma partição, isso só é útil para tabelas com múltiplas linhas por partição. Finalmente uma chave primária é uma sequência de colunas, sendo elas *partition key* ou, opcionalmente, *clustering key* [8].

3.3.4 Coluna

Coluna é o menor incremento de dados no banco de dados e possui três atributos: Nome, Valor e *Timestamp*. Como mostrado, o *Timestamp* será muito utilizado para garantir a consistência

Apesar de lembrar a estrutura do modelo relacional, o Cassandra possui divergências pontuais e que fazem diferença no gerenciamento dos dados. Assim como no modelo relacional, o banco *NoSQL* Cassandra também possui tabelas, que podem ser replicadas em mais de uma partição para garantir que não haverá perda de dados, e que estes estarão disponíveis quando preciso. Neste modelo os dados são *indexados* por uma tripla (linha, coluna e *timestamp*), onde linhas e colunas são identificadas por chaves e o *timestamp* permite diferenciar múltiplas versões de um mesmo dado. Vale ressaltar que operações de leitura e escrita são atômicas, ou seja, todos os valores associados a uma linha são considerados na execução destas operações, independentemente das colunas que estão sendo lidas ou escritas [1]. O modelo de dados do Cassandra é apresentado na Figura 3.5.

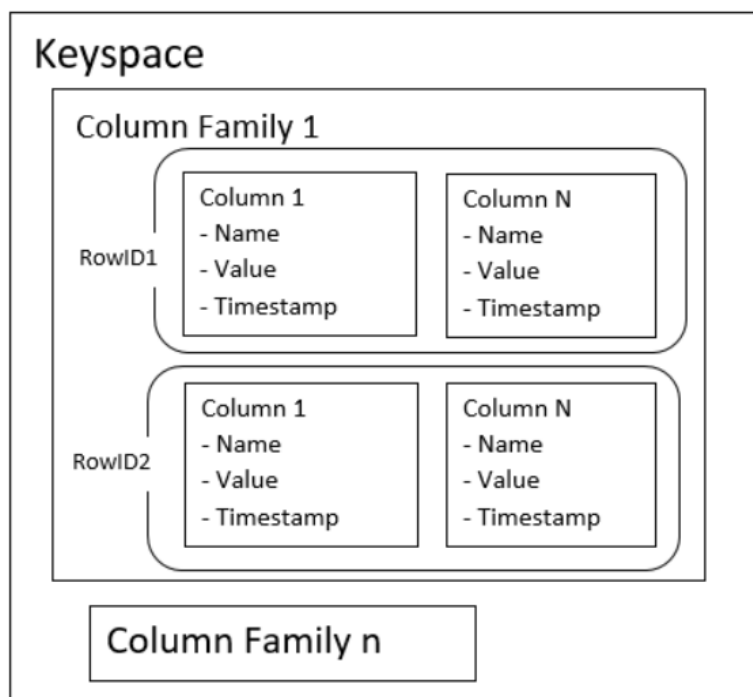


Figura 3.5: Modelo de Dados.

3.4 CQL

CQL é uma abreviação para *Cassandra Query Language* e é a linguagem utilizada para se comunicar com o Banco de Dados Cassandra, por meio dela é possível criar e deletar *keyspaces* e tabelas, inserir e consultar dados e muito mais. Essa linguagem é baseada no SQL, sendo assim possui muitas semelhanças com a mesma. Uma diferença crucial entre as duas linguagens é que CQL não aceita *Joins* ou *subqueries*, o que gera um grande impacto na modelagem dos dados. Cassandra busca utilizar apenas uma tabela por consulta. Por outro lado, o CQL possui funcionalidades como *clustering* e *collections*, que são utilizadas para buscar dados específicos ou buscar uma lista de dados, estas são especificadas para uma certa tabela, o que aumenta a eficiência na consulta [2].

Capítulo 4

Resultados

Neste capítulo é apresentado a comparação entre a administração dos sistemas gerenciadores de banco dados MySQL e Cassandra. A Seção 4.1 explica a análise comparativa realizada nessa pesquisa onde as atividades do DBA são analisadas entre dois SGBDs diferentes. É também falado sobre as tecnologias e configurações utilizadas e por fim na Seção 4.2 os resultados das análises são demonstrados, comparando as atividades do DBA para cada atividade selecionada.

4.1 Contextualização

Este trabalho é uma análise das diferenças quanto as atividades do DBA entre dois SGBDs diferentes, o MySQL baseado no modelo relacional, e o Cassandra do modelo NoSQL baseado em coluna. Como demonstrado nos capítulos anteriores, o modelo relacional e o NoSQL possuem diversas diferenças, e o objetivo desta pesquisa é entender o que isso reflete nas atividades do DBA por meio de uma análise das atividades do DBA para cada um dos SGBDs. Como o DBA possui um grande número de responsabilidades, utilizou-se uma amostra destas para esta pesquisa. As atividades escolhidas foram:

- Modelagem de Dados;
- Segurança e Auditoria;
- Backup e Recuperação.

Instalação

Para simular as ações do DBA em cada um dos SGBDs instalou-se sistemas gerenciadores de bancos de dados MySQL e Cassandra. O sistema operacional utilizado na análise foi o Linux, utilizando a distribuição Ubuntu, versão 14.04.3 LTS. Depois de uma análise sobre alguns sistemas gerenciadores de banco de dados do Modelo Relacional, foi escolhido o MySQL, devido ao fato dele ser um dos SGBD SQL mais populares [4]. As informações utilizadas quanto a estrutura dos SGBDs e dos comandos utilizados neles foram retiradas das documentações do Cassandra 3.x fornecida e mantida pela DataStax, e do MySQL 5.7 fornecida e mantida pela Oracle. A seguir será apresentado as configurações dos SGBDs utilizados.

Para esta análise foi utilizado o *MySQL Server Community* da versão 5.7.12, uma das versões mais recentes lançadas. O site do MySQL oferece uma fácil instalação do SGBD por meio do seu *download*, disponibilizado para as diversas plataformas, iOS, Windows e Linux. Na análise foi utilizado a versão para o Linux.

Para a instalação do Cassandra foi utilizado um pacote *opensource* fornecido pela Datastax, empresa que desenvolve e disponibiliza suporte para a utilização do SGBD Cassandra. Para realizar esta análise utilizou-se a versão do Cassandra 3.2.1, sua instalação foi feita através de linha de comando pelo terminal. Para utilização do Cassandra é necessário se ter o Java 7 ou 8.

A arquitetura do Cassandra pode ser moldada de acordo com a realidade da aplicação que a está utilizando. Para configuração do Cassandra, um arquivo importante é o *cassandra.yaml*, situado na pasta */etc/cassandra/*, ele possui as variáveis de configuração do banco de dados e a maioria delas possui um valor pré setado básico para a utilização do Cassandra. A utilização do Cassandra é normalmente feita de forma distribuída utilizando um grande número de nós, porém, neste trabalho, buscando apenas representar um sistema distribuído, criou-se um cluster básico com dois nós como mostrado na Figura 4.1, onde a máquina 1 e 2 representam cada um dos nós criados no cluster. Para isso foram necessárias duas máquinas com IPs diferentes e com o Cassandra instalado em cada uma. Tendo isso, é necessário parar o processo do Cassandra em cada uma das máquinas e então configurar as seguintes variáveis no arquivo *cassandra.yaml* em ambas máquinas:

- *cluster_name*: Nome do cluster criado;
- *seed_provider*: Escolhe-se uma das máquinas para ser *seed*. Como explicado no capítulo 3, o nó *seed* é utilizado apenas quando uma nova máquina entra no cluster para redistribuição dos dados;
- *listen_address*: IP que o outro nó do cluster deverá se conectar;
- *rpc_address*: Endereço para que clientes possam executar processos remotos.

Por fim o cluster funciona como programado ao inicializar novamente o Cassandra nas máquinas. É possível observar o cluster com os nós por meio do comando *nodetool.status* como demonstrado na Figura 4.2. A sigla UN presente a esquerda de cada máquina demonstra que o Cassandra está rodando normalmente em cada uma delas.

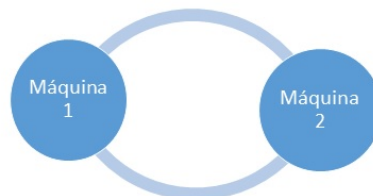


Figura 4.1: Imagem cluster utilizado.

```

hermano@hermano-Aspire-M5-583P:~$ nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address      Load       Tokens     Owns    Host ID                               Rack
UN  192.168.0.12  2.12 MB    256        ?       9832a773-202b-4b87-ae97-a9883c4284b7 rack1
UN  192.168.0.20  2.06 MB    256        ?       f21c4b25-4bfa-408b-9730-b110046e36dc rack1

```

Figura 4.2: Cluster criado.

Algumas das análises realizadas neste capítulo atualizam as configurações do Cassandra por meio deste arquivo, como será apresentado no decorrer do capítulo.

4.2 Análise Comparativa entre os SGBDs

Nesta seção é apresentada a análise realizada no projeto, onde se comparou as principais atividades de administração de banco de dados em dois ambientes diferentes: MySQL e Cassandra.

4.2.1 Modelo de Dados

Modelar os dados é a primeira parte do processo de desenvolvimento de um banco de dados e uma das mais importantes [24]. Algumas empresas maiores possuem empregados especializados apenas para esta etapa, mas em um grande número de casos, ela é realizada pelo DBA. Além disso, modelo de dados é um ponto de grande divergência entre o MySQL e o Cassandra, o que impacta a administração destes SGBDs, sendo assim é importante sua comparação. Esta etapa consiste em abstrair os dados utilizados na aplicação e representá-los em diagramas e tabelas com informações que possibilitarão a construção do banco de dados.

A modelagem dos dados para o MySQL possui um processo definido, com etapas e metodologias que permitem ao DBA, seguindo uma sequência de passos, criar um modelo conceitual, transformá-lo em lógico e por fim produzir o modelo físico [24]. Este *design*, tem como foco principal os relacionamentos entre as entidades, minimizar a redundância, e evitar duplicidade de dados [8]. Isso é atingido por meio da normalização dos dados utilizando as 3 formas normais propostas por Edgar F. Codd [10]. Nesta etapa não se leva em consideração as consultas realizadas pela aplicação, ou seja, como que a aplicação interage com o banco de dados. Isso acontece, pois, o SQL possui um conjunto de comandos como, *joins*, consultas agrupadas, funções de *data aggregation*, entre outros, que possibilitam uma grande interação com o banco de dados para se conseguir os dados necessários. De acordo com Chebotko et. al em [8] o processo de modelar dados para o modelo relacional é dirigido a dados, onde os padrões de acesso aos dados são apenas levados em consideração para otimizar as consultas mais frequentes.

A modelagem dirigida a dados não se adequa a realidade do Cassandra por diversos motivos. Primeiramente, o modelo de dados do Cassandra foi desenhado para obter o mínimo tempo de resposta, mesmo para aplicações complexas com grande volume de dados, e a necessidade de realizar relacionamentos pode ser um obstáculo. Segundo, o CQL, apesar de ser baseado no SQL, não suporta operações de *joins* ou agregação de dados, dificultando o relacionamento entre tabelas. Desta forma, para modelar o banco

de dados no Cassandra, é importante entender as consultas realizadas pela aplicação no banco de dados a fim de gerar tabelas mais adequadas e eficientes. Normalizar os dados sem observar essas consultas, como acontece para o MySQL, pode fazer com que buscas de dados se tornem ineficientes ou até mesmo não suportadas pelo CQL. O modelo de dados proposto pelo Cassandra possui um *design* que, utilizando acoplamento de dados e denormalização, permite realizar consultas complexas utilizando apenas uma tabela, eliminando o custo de relacionar diferentes tabelas para buscar dados corretos. Isso pode gerar dados duplicados no SGBD, o que não é visto como um problema uma vez que se acredita que espaço em disco é um recurso barato. Essa modelagem é dita como dirigida à consulta [8].

Como demonstrado, os conceitos existentes para modelagem de dados no SGBD Cassandra são muito diferentes daqueles utilizados para o MySQL. A modelagem dirigida a consulta e desnormalizada é uma quebra de paradigma para DBAs acostumados com a modelagem dirigida a dados e normalizada do modelo relacional. Isso aumenta consideravelmente a complexidade deste processo. É necessário desconstruir quase tudo que se sabe sobre a modelagem de dados utilizada no modelo relacional para aprender a mesma atividade no Cassandra.

Uma outra dificuldade encontrada foi a falta de um padrão bem definido para a modelagem dos dados no Cassandra. Enquanto para o modelo relacional essa metodologia é, após anos de estudo, bem definida, a comunidade da área de banco de dados Cassandra ainda está desenvolvendo metodologias para chegar a essa realidade, dado sua criação recente quando comparado com o modelo relacional. Uma pesquisa realizada pela Universidade de Wayne em 2015 descreve uma das primeiras metodologia para modelagem de dados no Cassandra e é utilizado em treinamentos realizados pela DataStax [8]. Este artigo foi utilizado como base para modelar o banco de dados no Cassandra nesta monografia.

Observadas as diferenças existentes entre o MySQL e o Cassandra quanto ao *design* do modelo de dados, analisaremos agora como é realizada a criação do banco de dados em ambos casos, desde sua modelagem conceitual até sua criação em SQL no MySQL ou CQL no Cassandra. Para facilitar a análise e torná-la mais real, a modelagem foi baseada em um banco de dados oferecido pelo MySQL, chamado *Employees Sample*¹, que possui aproximadamente 160MB de dados distribuído em 6 tabelas. Criou-se então um cenário hipotético para utilização deste banco de dados, assim como consultas hipotéticas para extrair dados deste cenário. Desta forma foi possível, seguir todos os passos da modelagem para o MySQL a fim de chegar ao resultado esperado, o banco de dados *Employees Sample*. Realizou-se também todos os passos da modelagem dos dados para o Cassandra, obtendo as tabelas necessárias para realização das consultas propostas. Desta forma o processo de modelagem dos dados assim como a criação das tabelas pode ser analisado.

- Cenário hipotético - Uma empresa com diversos departamentos onde funcionários podem estar trabalhando ou gerenciando um departamento em uma certa data. Os funcionários possuem nome, data de nascimento, gênero e dia que foram contratados. Além disso eles também possuem títulos e salários que podem mudar ao passar do tempo. Para uma data específica um funcionário pode apenas possuir um título e um salário assim como estar em apenas um departamento;

¹Banco de dados obtido em <https://dev.mysql.com/doc/employee/en/>

- Consultas hipotéticas no banco de dados ;
 - Buscar nome, identificador, salários e títulos, ordenados por ano, dos funcionários, que serão encontrados pelo nome;
 - Buscar, dado um ano, o quadro de funcionários de um departamento mostrando o título e o salário de cada um.

O processo de modelar esses dados será realizado em três etapas para os dois modelos:

- O modelo conceitual é a visualização geral dos dados e é independente de SGBD. Geralmente é representado pelo diagrama entidade-relacionamento, que descreve todas as entidades utilizadas, seus atributos e relacionamentos [24];
- O modelo lógico agrega informações ao modelo conceitual, buscando aproximar a representação de um banco de dados. Sua representação é geralmente em tabela e começa a se definir as colunas que serão utilizados no banco de dados assim como as chaves das entidades;
- O modelo físico implementa o modelo lógico para um banco de dados específico [24].

Modelo Conceitual

Essa etapa busca entender quais dados estarão envolvidos na aplicação e como estes se relacionam. Chebotko et al. em [8] descreve a importância de não apenas entender os dados utilizados pela aplicação, como acontece para o MySQL, mas também de entender como estes são acessados.

Para descrever melhor os dados utilizados na aplicação se construiu um diagrama entidade relacionamento o qual servirá tanto para o MySQL quanto para o Cassandra. Apesar de se construir um mesmo diagrama, este é utilizado de forma diferente nas próximas etapas, como demonstrado na Seção 4.2.1. Para aplicação proposta foi criado um diagrama entidade relacionamento com a notação proposta por Chen [9] por ser independente de tecnologia e não sofrer influência do modelo relacional. Ele possui 4 entidades: Funcionários, Departamentos, Salários e Títulos que possuem propriedades e se relacionam como mostrado na Figura 4.3.

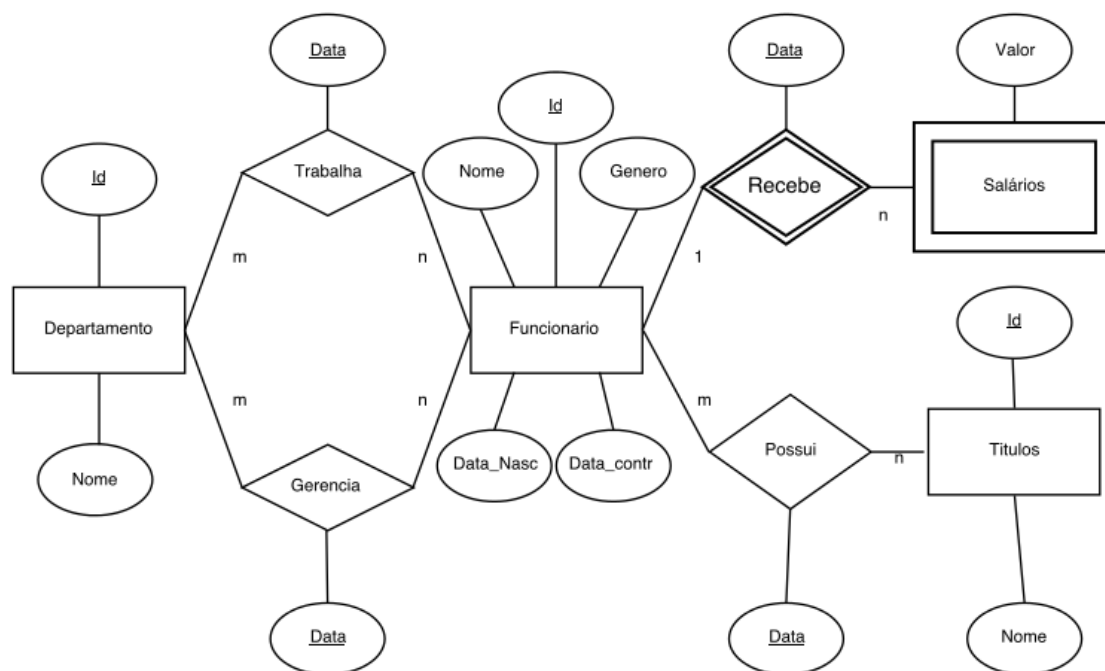


Figura 4.3: Diagrama Entidade Relacionamento para o cenário proposto.

Para a modelagem dos dados no MySQL esse diagrama seria o bastante para construção das tabelas normalizadas do modelo lógico. Porém, para o Cassandra, segundo Chebtko [8], é necessária mais uma etapa que busca entender como os dados são acessados pela aplicação a qual consiste em criar um diagrama do fluxo da aplicação que define padrões de acesso aos dados. Os padrões de acesso demonstram quais consultas a aplicação faz ao banco de dados e quais dados estas utilizam, possibilitando que as próximas etapas gerem tabelas mais adequadas para a realidade da aplicação. Criou-se então, para o cenário proposto, um fluxo de acesso aos dados para exemplificar esta etapa, onde as consultas Q1 e Q2, apresentadas a seguir, resultam em dados que são apresentados ao usuário.

- Q1: Buscar salários e títulos do funcionário dado um nome. Ordenar o resultado por ano;
- Q2: Buscar todos os funcionários com seus salários e títulos de um departamento em um ano.

Modelo Lógico

Nesta etapa a modelagem se diferencia bastante entre os modelos relacional e NoSQL baseado em colunas. Enquanto o MySQL utiliza as informações criadas no diagrama entidade relacionamento para criar tabelas e colunas, que são normalizadas a fim de criar os relacionamentos e identificações dos dados, o Cassandra utiliza os dois diagramas criados no modelo conceitual como forma de buscar entender melhor a aplicação e assim criar tabelas que tornaram as consultas que serão realizadas mais eficientes.

Em geral, para o mapeamento entre o modelo de entidade relacionamento e o modelo relacional, cria-se uma tabela para cada entidade, este é o caso para o MySQL. Diferentemente, no Cassandra deve ser criada uma tabela para cada consulta mapeada. A Figura 4.4 apresenta o modelo lógico construído para a aplicação proposta para o MySQL. Este foi construído utilizando as informações existentes no diagrama entidade relacionamento. Nesta etapa também são definidas as chaves de cada tabela, se utilizando de PK para chave primária e FK para chave estrangeira. Chaves primárias e estrangeiras são os atributos que identificam unicamente uma tabela, ou seja, duas linhas não possuem valores iguais para estes atributos, sendo chave primária um identificador próprio da tabela e chave estrangeira um identificador de outra tabela, utilizada para relacionar tabelas [24]. Criou-se uma tabela para cada entidade do modelo conceitual: Funcionario, Titulo, Salario e Departamento. Cada tabela possuindo os atributos também gerados no modelo conceitual. Como, no modelo conceitual, Funcionário possui dois relacionamentos M para N com Departamento, gerenciando e trabalhando, criou-se então mais duas tabelas que relacionam os identificadores de Funcionario com os identificadores de Departamento, sendo estas chaves destas tabelas. Tanto Salário é uma entidade fraca de Funcionario, ou seja, dependem deste para existir, desta forma o identificador de Funcionario é uma chave estrangeira de Salario, por fim Titulo no modelo conceitual possui um relacionamento M para N com Funcionario, porém na sua transição para o modelo lógico criou-se apenas uma tabela de Títulos que se relaciona com Funcionarios onde serão listados os funcionários com a título que possuem e a data em que receberam esse título, isso para que se adeque melhor ao modelo físico final proposto. Nesta etapa, para o MySQL, também são atribuídos os tipos aos atributos.

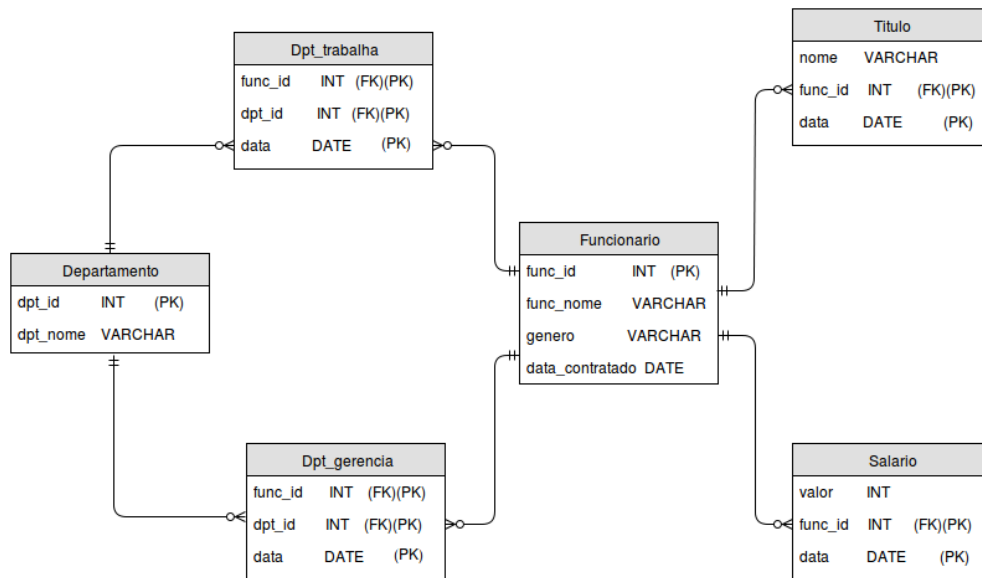


Figura 4.4: Modelo lógico para o MySQL.

Chebotko et al. em [8], apresenta 4 princípios e 5 regras que baseiam a criação do modelo lógico. Os 4 princípios apresentados no artigo, são formas de se estudar os dados criados no modelo conceitual de forma que facilite a criação do modelo lógico para o Cassandra. Estes princípios são:

- Conheça os dados: um maior entendimento do dado é capturado pelo modelo conceitual, principalmente pelo diagrama entidade relacionamento. As entidades, relacionamentos e atributos criados não definem apenas quais dados devem ser armazenados, mas também como estes devem ser organizados estabelecendo chaves e cardinalidades;
- Conheça as consultas: as consultas são analisadas pelo modelo de fluxo da aplicação, e seu correto entendimento afeta diretamente no *design* das tabelas, da forma que, se uma consulta mudar, provavelmente uma tabela também mudará;
- Acoplamento de dados: este princípio se refere à técnica que junta múltiplas entidades baseado em uma condição. Por exemplo, buscar funcionários por departamento junta as entidades funcionário e departamento. Isso facilita atingir o objetivo de utilizar apenas uma tabela por consulta;
- Duplicação de dados: uma prática comum utilizada no Cassandra é duplicar os dados sobre múltiplas tabelas, partições ou linhas. Isto é necessário para que o SGBD suporte diferentes consultas para um mesmo dado.

As 5 regras apresentadas no artigo propõem passos para a criação do modelo lógico a partir do modelo conceitual, elas devem ser seguidas para cada consulta do modelo conceitual. A Figura 4.5 demonstra uma tabela sendo montada para a consulta Q1 criada no modelo conceitual, seguindo as regras da metodologia. Este modelo é representado utilizando a notação proposta pelo próprio Chebotko, aonde K denota uma chave de partição e C uma chave do cluster (*clustering key*), e a seta demonstra se a ordenação da consulta é feita de forma ascendente ou descendente. As regras são:

- Regra 1 - Entidades e relacionamentos são mapeados em tabelas e os atributos que descrevem entidades e relacionamentos no modelo conceitual devem ser preservados como colunas no modelo lógico. Como pode ser observado na Figura 4.5, esta regra é aplicada gerando a tabela `func_by_nome` com as colunas `id`, `nome`, `salario`, `titulo` e `data`. Essa tabela juntou dados das entidades `Funcionario`, `Titulos` e `Salarios` em uma única tabela do Cassandra para representar a consulta Q1;
- Regra 2 - Os atributos utilizados como busca na consulta são mapeados como chaves primárias da tabela. Esses atributos devem incluir todas as chaves de partição. Como no exemplo se busca os funcionários por nome, então este atributo é chave de partição da tabela, denotado pela letra K como pode ser observado na Figura 4.5;
- Regra 3 - Atributos utilizados de forma secundária na busca são mapeados como *clustering key* na tabela. Como mostrado na aplicação da regra 3 na Figura 4.5, o resultado da consulta deve ser ordenado por ano, assim a data é uma *clustering key* denotado pela letra C com uma seta ao lado, demonstrando se a ordenação será ascendente ou descendente, por padrão é criada como ascendente;
- Regra 4 - As chaves de *cluster* definem o ordenamento que a consulta irá retornar, a regra 4 propõe que esta seja definida conforme a consulta criada no modelo conceitual. Para Q1 ordena-se o retorno por ano de forma descendente, do mais atual para o mais antigo, sendo assim, como pode ser observado na Figura 4.5 a seta do atributo ano é invertida;

- Regra 5 - Atributos de entidades que são chaves no modelo conceitual, devem ser mapeados como chaves no modelo lógico para os identificar unicamente na tabela. A Figura 4.5 mostra que para a tabela `func_by_nome` o atributo `id` é chave primária da entidade funcionário, e por isso é definido como *clustering key* para esta tabela.

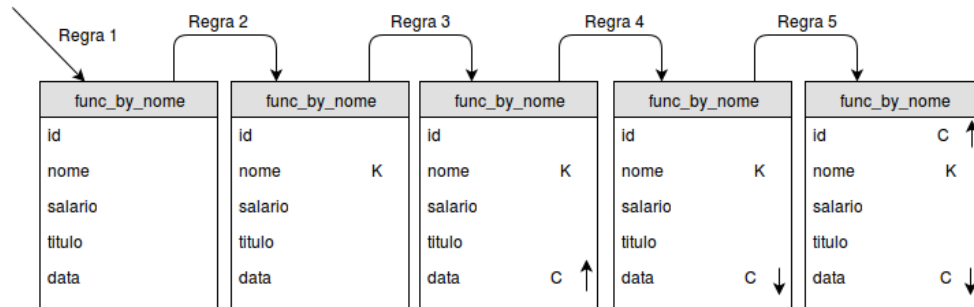


Figura 4.5: Modelo lógico para o Cassandra demonstrando todas as regras sendo aplicadas.

Deseja-se ao final que cada consulta gerada no modelo conceitual seja atendida com apenas uma tabela. A Figura 4.6 mostra o modelo lógico completo para a aplicação para o Cassandra.

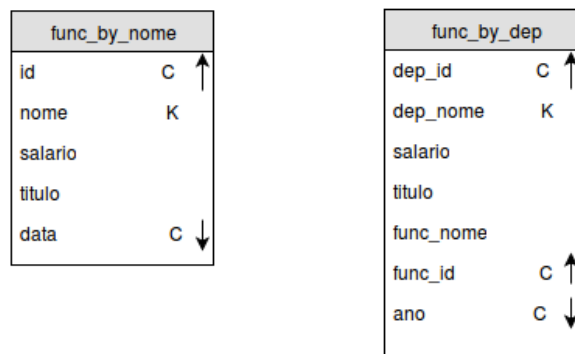


Figura 4.6: Modelo lógico completo utilizando a notação Chebotko.

A construção do modelo lógico para o Cassandra requer um maior conhecimento sobre a aplicação onde é necessário um estudo sobre cada dado e como estes serão acessados. Este fato torna o processo mais complexo e lento, em troca de um menor tempo na hora de consultar os dados. Por outro lado, para o MySQL a criação do modelo lógico é mais simples e pode ser realizada em menos passos sem a necessidade de uma análise profunda das consultas mapeando o diagrama entidade-relacionamento em tabelas e normalizando se necessário.

Modelo Físico

Apesar de a modelagem no MySQL adicionar os tipos dos dados no modelo lógico, sendo seu modelo físico a real construção do banco de dados, Chebotko em seu artigo inclui

os tipos dos atributos apenas no modelo físico. Além disso, esta etapa é utilizada para realizar uma análise sobre o modelo lógico em busca de melhorias em eficiência e economia de espaço. Apesar de o Cassandra propor que espaço em disco é um recurso barato, e que por vezes dados podem ser duplicados, isto deve ser realizado em troca de um ganho na eficiência das consultas realizadas pela aplicação. Estimativas como tamanho da partição ou quantidade de dados duplicados, podem ser utilizadas como indicadores da eficiência do banco. Algumas técnicas que podem ser aplicadas caso problemas sejam encontrados na construção são: separação de partição e agregação de dados [8]. Percebe-se novamente que a modelagem realizada no Cassandra busca adequar o banco o máximo possível para a aplicação, enquanto o modelo relacional existe um maior cuidado com a forma que os dados estarão armazenados o que torna a modelagem para o Cassandra mais complexa. O modelo físico final do Cassandra é apresentado na Figura 4.7.

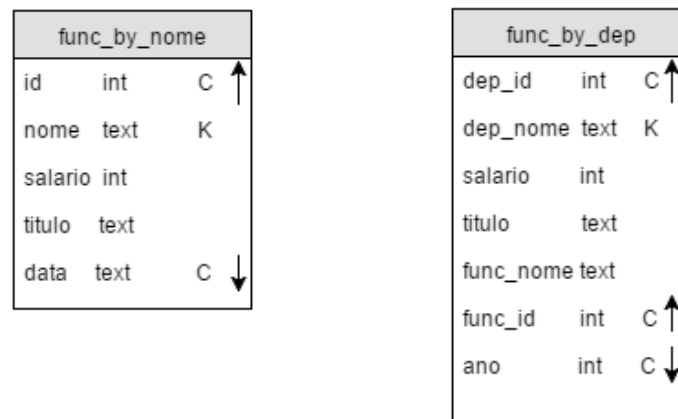


Figura 4.7: Modelo físico Cassandra.

Uma vez que o modelo físico é aquele construído no banco de dados, esta etapa também englobará os comandos CQL e SQL para a criação das tabelas. Como Cassandra, com o CQL, e o MySQL são baseados no SQL as estruturas são semelhantes, e os comandos são parecidos. As Figuras 4.8 e 4.9 demonstram os comandos realizados no CQL e SQL respectivamente para a criação de uma tabela do modelo proposto para o Cassandra e do modelo proposto para o MySQL.

```
CREATE TABLE funcionarios (
    id int,
    nome text,
    titulo text,
    salario int,
    data text,
    PRIMARY KEY(nome, id, data))
WITH CLUSTERING ORDER BY (id asc, data desc);
```

Figura 4.8: Comando CQL para criação de tabela.

```
CREATE TABLE funcionario (  
func_id int,  
func_nome varchar(35),  
genero char,  
data_contratado date);
```

Figura 4.9: Comando SQL para criação de tabela.

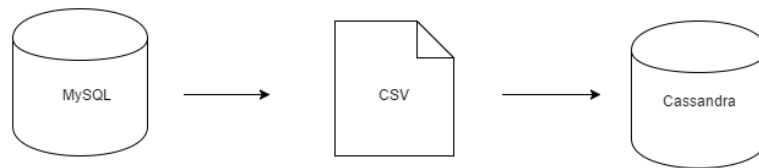


Figura 4.10: Importação de dados do MySQL para o Cassandra.

Como esperado, o modelo físico criado para o MySQL é igual ao proposto no *Employees Sample* oferecido pelo MySQL, desta forma foi possível importar sua base de dados para o SGBD Cassandra local criado. A Figura 4.10 demonstra o processo necessário para importar estes dados do MySQL para o Cassandra. Primeiramente, foi necessário uma consulta que selecionasse, no MySQL, os dados necessários para cada tabela criada no Cassandra e os colocasse em um arquivo CSV. Esta consulta necessitou juntar as colunas de primeiro nome e último nome existente no MySQL em apenas uma coluna, chamada nome. Para a data, utilizou-se apenas a data de início. Por fim, este pode então ser importado para o Cassandra por meio do comando *COPY FROM*, que insere dados de um arquivo em uma tabela do banco de dados. A Figura 4.11 mostra a consulta realizada no MySQL para buscar os dados para a tabela *func_by_id* e a Figura 4.12 mostra o comando realizado no Cassandra para copiar os dados do arquivo *MySQLToCassandra.csv* criado para a tabela do banco de dados *func_by_id*.

```

SELECT  n.emp_no,
        n.nome,
        n.salary,
        IFNULL(t.title, ''),
        n.from_date
FROM (
    SELECT  s.salary,
            s.from_date,
            s.to_date,
            s.emp_no,
            concat(e.first_name, ' ', e.last_name) AS nome
    FROM employees AS e
    JOIN salaries AS s
    ON e.emp_no = s.emp_no
) AS n
LEFT JOIN titles AS t
    ON n.from_date <= t.from_date
    AND n.to_date >= t.from_date
    AND n.emp_no = t.emp_no
INTO OUTFILE '/tmp/mysqltocassandra'
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\r\n';

```

Figura 4.11: Consulta em SQL para gerar arquivo CSV.

```

cassandra@cqlsh:testeccluster> COPY funcionarios (id, nome, salario, titulo, data) FROM '/tmp/mysqltocassandra';
Using 3 child processes

Starting copy of testeccluster.funcionarios with columns ['id', 'nome', 'salario', 'titulo', 'data'].
Processed: 2844047 rows; Rate: 51543 rows/s; Avg. rate: 8416 rows/s
2844047 rows imported from 1 files in 337.933 seconds (0 skipped).

```

Figura 4.12: Comando para copiar dados do arquivo CSV para o CQL.

Para esta etapa da análise, para o Cassandra utilizou-se um *cluster* com dois nós como demonstrado no início do capítulo, e um *keyspace* replicado entre eles com (*replication_factor* = 2). Por fim as tabelas foram criadas dentro deste *keyspace*. As Figura 4.13 e 4.14 mostram as consultas realizadas em SQL e CQL respectivamente para a consulta propostas no início da modelagem, que foi: Buscar nome, identificador, salários e títulos, ordenados por ano, dos funcionários, que serão encontrados pelo nome.

```
mysql> SELECT n.emp_no, n.nome, n.salary, t.title, n.from_date
      FROM (
        SELECT  s.salary,
                s.from_date,
                s.to_date,
                s.emp_no,
                concat(e.first_name, ' ', e.last_name) AS nome
        FROM employees AS e
        JOIN salaries AS s
        ON e.emp_no = s.emp_no
        AND e.first_name = 'Jahangir'
        AND e.last_name = 'Speer') AS n
LEFT JOIN titles AS t
ON n.from_date <= t.from_date
AND n.to_date >= t.from_date
AND n.emp_no = t.emp_no
ORDER BY n.from_date desc;
```

emp_no	nome	salary	title	from_date
19999	Jahangir Speer	85689	NULL	2001-09-26
19999	Jahangir Speer	82375	NULL	2000-09-26
19999	Jahangir Speer	81688	NULL	1999-09-27
19999	Jahangir Speer	80554	Senior Engineer	1998-09-27
19999	Jahangir Speer	77775	NULL	1997-09-27
19999	Jahangir Speer	74254	NULL	1996-09-27
19999	Jahangir Speer	71867	NULL	1995-09-28
19999	Jahangir Speer	71793	NULL	1994-09-28
19999	Jahangir Speer	67398	NULL	1993-09-28
19999	Jahangir Speer	63362	NULL	1992-09-28
19999	Jahangir Speer	59430	NULL	1991-09-29
19999	Jahangir Speer	56486	NULL	1990-09-29
19999	Jahangir Speer	52813	Engineer	1989-09-29

13 rows in set (0.22 sec)

Figura 4.13: Consulta SQL.

```
cassandra@cqlsh:testecoluster> select * from funcionarios where nome = 'Jahangir Speer';
```

nome	id	data	salario	titulo
Jahangir Speer	19999	2001-09-26	85689	null
Jahangir Speer	19999	2000-09-26	82375	null
Jahangir Speer	19999	1999-09-27	81688	null
Jahangir Speer	19999	1998-09-27	80554	Senior Engineer
Jahangir Speer	19999	1997-09-27	77775	null
Jahangir Speer	19999	1996-09-27	74254	null
Jahangir Speer	19999	1995-09-28	71867	null
Jahangir Speer	19999	1994-09-28	71793	null
Jahangir Speer	19999	1993-09-28	67398	null
Jahangir Speer	19999	1992-09-28	63362	null
Jahangir Speer	19999	1991-09-29	59430	null
Jahangir Speer	19999	1990-09-29	56486	null
Jahangir Speer	19999	1989-09-29	52813	Engineer

(13 rows)

Figura 4.14: Consulta CQL.

Pode se observar a diferença de complexidade existente nas buscas, onde a consulta realizada em CQL é mais simples que a realizada em SQL. É interessante analisar que a simplicidade encontrada no Cassandra se dá devido ao fato de as tabelas deste estarem preparadas para uma consulta específica, enquanto o MySQL necessita se adaptar, por meio de uma consulta mais complexa, para encontrar os mesmos dados.

Observou-se também as limitações existente no CQL para realizar consultas. Como explicado, o Cassandra possui uma arquitetura preparada para ser escalável, e desta forma organiza os dados de forma que possam ser encontrados mais rápido e facilmente, então as consultas devem ser realizadas para que se aproveite o máximo desta arquitetura, e o CQL impõe limitações para que isto aconteça.

Um exemplo disso, observado na análise, foi a utilização das chaves de partição na tabela *func_by_id*. A modelagem descrita por Chebotko et al. não específica na regra 5 se o atributo chave do modelo conceitual se torna uma *partition key* ou uma *clustering key* no modelo lógico, e inicialmente o *id* do *funcionário* foi criado como uma chave de partição na tabela *func_by_id*. Entretanto, o CQL exige que todas as chaves de partição estejam presentes nas consultas e que as operações executadas com estas chaves sejam apenas de igualdade (= ou *IN*), retornando um erro caso contrário. Existe uma forma de contornar este problema por meio do uso de índices, que tem o objetivo de facilitar a busca quando muitas linhas possuem um mesmo valor, porém esta não foi utilizada neste trabalho. Sendo assim, sem utilizar índices, tornou-se inviável que o *id* do *funcionário* fosse uma chave de partição, sendo por fim, no modelo lógico final, uma chave de *cluster*. O Cassandra exige isso, pois ele produz um *hash* das chaves de partição e realiza a busca do dado por meio deste, isso torna a busca do valor mais eficiente. Por fim, outro fato observado foi o fato de a pré determinação, na criação da tabela, da ordenação de retorno da consulta por ordem decrescente de data, simplificar a consulta final em CQL.

4.2.2 Segurança

Os dados armazenados em um banco de dados são de extrema importância para o funcionamento de uma empresa e, por isso, os dados devem ser protegidos. Além disso, é necessário garantir que esses dados sejam utilizados de maneira correta pela pessoa correta. Sendo assim, uma função importante do DBA é controlar quem pode acessar os dados de um banco de dados, quais dados estes podem acessar e o que podem fazer com eles. Por fim ainda devem manter um controle de como os usuários utilizaram este banco de dados para encontrar a causa de possíveis problemas. Essa tarefa é crítica para o DBA e demanda um esforço contínuo. Essa etapa possui semelhanças para os dois SGBDs, com diferenças pontuais que são demonstradas ao longo desta seção. Ambos os modelos garantem a segurança em duas etapas:

- Autenticação: Garantir que o banco de dados seja acessado apenas por usuários com permissão;
- Autorização: Limitar as ações feitas pelos usuários no banco de dados, podendo ser controladas as ações que estes podem realizar assim como com quais dados elas podem ser aplicadas.

Cada uma dessas etapas será analisada separadamente a seguir.

Autenticação

Para garantir que apenas usuários com permissão acessem ao SGBD, ambos modelos permitem o acesso ao sistema por meio de usuários e senhas. Sendo assim, ao tentar se conectar o usuário será liberado apenas se as informações de autenticação forem as mesmas que foram previamente cadastradas. Tanto em SQL quanto em CQL, são utilizados comandos para criar os usuários com suas senhas. Os dois modelos guardam as informações de usuários e suas senhas encriptadas em tabelas no banco de dados. O Cassandra as guarda na tabela *credentials* e o MySQL na tabela *user*.

Por padrão, o MySQL vem com a autenticação funcionando e no momento da sua instalação no Linux é criado um usuário administrador com as credenciais do *root* do sistema operacional. Essa configuração pode ser sobrescrita caso necessário apenas alterando o usuário. Por outro lado, no Cassandra, o padrão é possuir a autenticação desativada, com livre acesso ao banco por qualquer usuário com acesso à máquina. Para configurar a autenticação é necessário editar o arquivo de configuração do banco, *cassandra.yaml*, e alterar a variável *authenticator* de *AllowAllAuthenticator* para *PasswordAuthenticator*, a configuração funcionará após o banco ser reiniciando. Por padrão o Cassandra cria o usuário “cassandra” com senha “cassandra” e permissão de administrador (*superuser*) para que o banco possa ser acessado e configurado. Essa configuração também pode ser atualizada alterando o usuário padrão.

Os comandos para criar um usuário novo no banco de dados são muito semelhantes nos modelos. Os dois utilizam o comando *CREATE USER* para criar o usuário e ambos possuem a opção de incluir o comando *IF NOT EXISTS*, para criar o usuário apenas se ele não existir. A sintaxe para configurar a senha muda, sendo ela *IDENTIFIED BY* para o MySQL e *WITH PASSWORD* para o Cassandra como mostrado nas Figuras 4.15 e 4.16. O MySQL ainda possui mais algumas opções que podem ser utilizadas na criação do usuário como expiração de senha e bloqueio de usuário, já o Cassandra tem a opção de criar um usuário como “superusuário”.

```
mysql> CREATE USER 'usuarioteste' IDENTIFIED BY '123'
Query OK, 0 rows affected (0.00 sec)
mysql> exit;
Bye
hermano@hermano-Aspire-M5-583P:~$ mysql -u usuarioteste -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 45
Server version: 5.5.49-0ubuntu0.14.04.1 (Ubuntu)
```

Figura 4.15: Criação de usuário em SQL.

```
cassandra@cqlsh> CREATE USER usuarioteste WITH PASSWORD '123';
cassandra@cqlsh> exit;

hermano@hermano-Aspire-M5-583P:~$ cqlsh 192.168.0.12 -u usuarioteste -p 123

Connected to ClusterTG at 192.168.0.12:9042.
[cqlsh 5.0.1 | Cassandra 3.2.1 | CQL spec 3.4.0 | Native protocol v4]
Use HELP for help.
```

Figura 4.16: Criação de usuário em CQL.

Enquanto o Cassandra faz a validação do usuário por meio de um nome de usuário e uma senha, o MySQL também associa um *host* a cada usuário como demonstrado na Figura 4.17. É possível que um usuário acesse de qualquer *host* caso não se queira ter essa limitação. Isso garante um poder maior ao DBA sobre os usuários.

```
mysql> CREATE USER 'usuarioteste'@'192.168.0.20' IDENTIFIED BY '123'  
Query OK, 0 rows affected (0.00 sec)
```

Figura 4.17: Criação de usuário em SQL com host.

Para deletar um usuário também é utilizado o mesmo comando: *DROP USER*. A forma para se listar os usuários criados no banco de dados, se diferencia entre os dois modelos. Enquanto para o MySQL é necessário realizar uma consulta à tabela de usuários, o Cassandra fornece o comando *LIST USERS* para fazer isso.

A forma como os bancos lidam com a autenticação de usuários é muito semelhante, inclusive nos comandos utilizados em SQL e CQL e na forma como os usuários criados são guardados. As diferenças se apresentam na definição padrão do sistema, onde para o Cassandra é necessário alterar a configuração inicial para autenticar os usuários e nas opções adicionais existentes na criação do usuário. Para a autorização as diferenças são mais aparentes.

Autorização

Uma vez que o usuário esteja autenticado, ou seja, este possui acesso ao banco e se tem conhecimento de quem ele é, pode se limitar os dados que estes podem acessar assim como os comandos que podem realizar por meio de privilégios. Privilégios são permissões dadas aos usuários de realizarem certas atividades dentro do banco de dados. Inicialmente os usuários são criados sem nenhuma permissão em ambos modelos e os comandos utilizados para conceder ou retirar as permissões são iguais tanto em SQL e CQL sendo elas *GRANT* e *REVOKE* respectivamente. Os dois modelos também permitem definir quais bancos de dados ou, sendo mais específico, quais tabelas um usuário pode ter acesso, assim como quais comandos este pode executar. Por exemplo, é possível permitir que um usuário apenas possa listar os dados de uma tabela de um banco de dados, mas não possa nem editar esses dados nem listar dados de uma outra tabela.

As diferenças começam quanto a granularidade dessas permissões onde a lista de permissões possíveis para o MySQL é bem mais extensa que a existente para o Cassandra o qual engloba, em um mesmo privilégio, um grande número de atividades possíveis. Enquanto o MySQL possui um maior número de privilégios autorizando, cada um, atividades mais específicas dentro do SGBD. O MySQL oferece 32 possíveis privilégios que podem ser concedidos aos usuários como pode ser visto na Tabela 4.1, enquanto para o Cassandra oferece apenas 7 que são: *ALL*, *ALTER*, *AUTHORIZE*, *CREATE*, *DROP*, *MODIFY* e *SELECT*. Isso oferece um poder muito maior ao DBA no MySQL, que pode configurar até o nível de qual coluna em uma tabela o usuário pode executar um comando. Permissões para comandos sobre o sistema também podem ser configuradas como é o exemplo do privilégio *SHUTDOWN*, algo que não existe no Cassandra. O Cassandra apresenta um controle não tão restrito, onde um usuário sem nenhuma permissão pode acessar o banco

Tabela 4.1: Tabela privilégios MySQL.

CREATE	DROP	GRANT OPTIONS
LOCK TABLES	REFERENCE	EVENT
ALTER	DELETE	INDEX
INSERT	SELECT	UPDATE
CREATE TEMPORARY TABLE	TRIGGER	CREATE VIEW
SHOW VIEW	ALTER ROUTINE	CREATE ROUTINE
EXECUTE	FILE	CREATE TABLESPACE
CREATE USER	PROCESS	PROXY
RELOAD	REPLICATION CLIENT	REPLICATION SLAVE
SHOW DATABASE	SHUTDOWN	SUPER
ALL	USAGE	

de dados: No MySQL este não consegue ver nenhum banco de dados existente no sistema, assim como nenhuma tabela. No Cassandra, este consegue ver os Keyspaces existentes assim como as tabelas criadas dentro deles, a permissão nega o acesso apenas quando este tenta interagir com qualquer dado.

Por outro lado, o Cassandra possibilita uma maneira mais flexível e fácil de controlar os privilégios que cada usuário possui. Desde a versão 2.2 o controle de acesso é baseado em *ROLES*. Esse esquema permite que *roles* sejam criados e que privilégios sejam atribuídos a eles ou retirados deles assim como acontece com usuário, a diferença é que os *roles* podem ser concedidos uns aos outros, assim como também pode ser concedido a um usuário. Um usuário também é analisado com um *role* pelo Cassandra, sendo assim é possível que se conceda todas as permissões de um usuário para outro facilmente. É possível listar todos os *roles* cadastrados por meio do comando *LIST ROLES*, como explicado os usuários cadastrados também aparecem nesta tabela, porém com a coluna *login* como *true*, isso é o que diferencia para o Cassandra um *role* de um usuário, a Figura 4.18 mostra a criação de um *role* assim como a listagem deste, é possível observar que os usuários do sistema também são listados, porém a coluna *login* tem valor *true*.

```
cassandra@cqlsh> CREATE ROLE roleteste;
cassandra@cqlsh> List ROLES;
```

role	super	login	options
cassandra	True	True	{}
roleteste	False	False	{}
usuarioteste	False	True	{}

Figura 4.18: Utilização do *role* no Cassandra.

Os *roles* podem ser vistos como um grupo de privilégios que buscam facilitar a administração dos usuários realizada pelo DBA que não tem a necessidade de gerenciar todos os privilégios para cada usuário, podendo então controlar grupos de privilégios.

É possível observar com a comparação realizada que o DBA no banco de dados Cassandra, por meio do recurso de *roles* pode ter facilidade para gerenciar os privilégios dos

usuários, porém o MySQL apresenta uma lista maior e mais completa de permissões que podem ser dadas ao usuário, garantindo um maior controle.

Auditoria

Auditoria é gerenciar as atividades que estão sendo executadas pelos usuários em um banco de dados, buscando saber quem fez o que no SGBD. Isso ajuda a descobrir o motivo de possíveis erros que aconteçam, além de manter o controle do sistema.

No Cassandra essa funcionalidade é fornecida pela *DataStax* e pode ser encontrada apenas na versão paga *DataStax Enterprise*. Existem duas opções, um *log* onde as atividades são escritas sequencialmente, ou em uma tabela. A segunda opção é mais viável para o caso de bancos de dados grandes, pois facilita a busca. A auditoria funciona por nó, então para se obter informações do banco de dados por completo, é necessário habilitar essa funcionalidade em todos os nós.

Para o MySQL, utilizando o *MySQL Enterprise Audit* isso funciona da mesma forma, porém as atividades serão escritas apenas em *log*. É possível selecionar quais as atividades que são escritas assim como quais partes do banco de dados são monitoradas, por meio das variáveis de sistema que começam com *audit*, por exemplo *audit log policy*. No entanto essa funcionalidade também é encontrada apenas na versão paga, mas ela utiliza da API *audit* da MySQL a qual pode ser utilizada, então caso não se deseje utilizar a versão paga é possível criar um *plugin* utilizando a API.

Estas são as ferramentas existentes para o Cassandra e para o MySQL, porém, nos dois SGBDs também é possível se construir uma estrutura para auditar o servidor, por meio de *triggers* e *procedures*. Desta forma é possível a partir de uma tabela criada, populá-la de maneira customizada podendo se adequar a realidade da aplicação. Esta maneira de se auditar, pode ser realizada nos dois modelos e consiste em utilizar *triggers* que são funções que são disparadas mediante uma ação.

4.2.3 Backup e Recuperação

Uma função muito importante de um DBA é garantir uma cópia de segurança dos dados que foram armazenados e guardados do sistema conhecido como *backup*, para que em caso de alguma perda ou algum problema com os dados originais, esses possam ser restaurados ou recuperados, essa parte é conhecida como recuperação.

Essa tarefa de *backup* e recuperação de dados é uma tarefa do DBA, de extrema importância porque se houver algum problema ou perda dos dados do sistema, o administrador do banco de dados precisa estar preparado para recuperar ou restaurar esses dados.

Sendo assim temos que tanto o modelo relacional quanto o banco de dados Cassandra apresentam formas de realizar o *backup* e a recuperação dos dados, os dois possuem têm ferramentas para isso, que se assemelham em alguns aspectos e se diferenciam em outros. Todos esses pontos serão tratados ao longo desta seção.

Backup

Para realização de *backups* o MySQL oferece o comando *mysqldump*, que já é disponível na instalação do SGBD. Ele funciona produzindo um conjunto de instruções em SQL que podem ser executados para reproduzir o banco de dados original, com suas tabelas,

relações e dados definidos, assim como o original. Pode se fazer um *dump* em um ou mais banco de dados MySQL ou transferir para outro SQL *server*. Por padrão o comando funciona gerando arquivos no formato *SQL-format dump*, e escreve as informações como instruções SQL, porém, também é possível que o comando gere saídas no formato CSV, outro delimitador de texto ou ainda no formato XML.

O *mysqldump* já vem no *package* de ferramentas do *MySQL Server*. Para sua utilização, usa-se linha de comando do Linux, utilizando o nome do comando e seus parâmetros da seguinte maneira: *mysqldump [parâmetros] > nome arquivo*. Essa é a forma para se utilizar o *mysqldump*, dando a chamada do comando, os argumentos que serão utilizados e o nome do arquivo onde serão armazenadas as instruções SQL, sendo gerado assim um arquivo *.sql*. A Figura 4.19 é um exemplo de como o *backup* do banco de dados é gerado.

```
mysqldump -u root -p senha dbTeste > dbSaida.sql
```

Figura 4.19: Comando Backup MySQL.

Na Figura 4.19 a *flag* *-u root* assim como *-p senha*, servem para autenticar se o usuário tem permissão para realizar *backup* do banco de dados. *dbTeste* é o banco de dados que está no *MySQL Server* que se quer gerar o *backup* e *dbSaida.sql* é o nome do arquivo *.sql* que será gerado após a execução do comando.

Existem ainda outras opções de argumentos que podem ser utilizados neste comando que podem ser utilizados para facilitar a atividade do DBA. É possível realizar o *backup* de todos os bancos de dados do *MySQL Server*, ou de somente alguns deles, ou também pode se realizar uma cópia de segurança das tabelas de um banco de dados específico. Um outro comando do *MySQL Server* para realização do *backup* é o comando *mysqlpump*, o qual já vem nas *packages* do *MySQL* e funciona de forma semelhante ao *mysqldump*. Por fim, para o MySQL, também é possível realizar backups por meio do MySQL workbench com uma interface gráfica que busca facilitar este processo.

Já no Cassandra, o processo de *backup* ocorre através do *snapshot* de todo disco de arquivos de dados conhecidos como *SSTable files*, que fica armazenado no diretório de dados. O *snapshot* é um comando do Cassandra, que funciona descarregando tudo que está escrito na memória para o disco, fazendo um *hard link* com a *SSTable files* para cada *keyspace*. É necessário muito espaço em disco no nó para acomodação dos *snapshots* tirados dos arquivos de dados, uma vez que os arquivos mais antigos não são deletados, diferente do que ocorre no MySQL, onde cada vez que o comando *mysqldump* é executado com a saída com mesmo nome de um arquivo existente, este é sobrescrito.

Semelhante ao *mysqldump* do MySQL, com o Cassandra é possível tirar *snapshots* de todos os *keyspaces*, de apenas um único *keyspace* ou também de uma única tabela de todo sistema. Existe ainda a possibilidade de tirar um *snapshot* de um *cluster* inteiro usando a ferramenta *parallel ssh tool*, que é um programa para executar *ssh* em paralelo a um número de *hosts*.

Uma funcionalidade interessante apresentada pelo Cassandra na realização de *backup* é a realização de *backups* incrementais, onde após um primeiro *snapshot* ser tirado, armazenando todo o banco de dados, os próximos guardam apenas as diferenças encontradas neste, economizando espaço. O *mysqldump* do MySQL não oferece esta opção, sendo isso possível apenas por meio de *scripts*. Para bancos de dados escaláveis, esta funcionali-

dade torna-se essencial. Os *snapshots* são tirados por nó, utilizando o comando *nodetool snapshot*. Para se tirar um *snapshot* global utiliza-se esse comando com o *parallel ssh utility*.

`./nodetool snapshot -t nomeSnapshot nomeKeyspace`

Figura 4.20: Comando Backup Cassandra.

Na Figura 4.20, o comando *snapshot* é utilizado junto com algumas *flags*, o *-t nomeSnapshot nomeKeyspace* serve para determinar o nome do *snapshot* e o nome do *keyspace* o qual está sendo feito o *backup*. Algumas outras *flags* podem ser utilizadas, para determinar o *host*, a porta, a senha, o usuário, a coluna de família.

Recuperação

Essa tarefa também é de extrema importância para o DBA, porque precisa ser bem feita para que se tenha uma recuperação total e correta dos dados. No *MySQL* a recuperação dos dados é feita da seguinte forma, com a geração do arquivo *.sql*, só precisa executar esse arquivo pelo *MySQL* mesmo e o banco de dados será montado, juntamente com os dados que foram salvos pelo *backup*. Porém existem alguns detalhes que devem ser observados. Para realizar essa recuperação, se utiliza linha de comando, primeiramente verifica-se se o banco de dados que queremos utilizar está criado ou não. Caso este não tenha sido criado, se utiliza o *CREATE DATABASE db* para criá-lo, e uma vez dentro do banco de dados por meio do comando *USE db*, é possível recuperar o *backup* com o comando *source db.sql* que utiliza o arquivo *.sql* para pegar as instruções SQL que estavam contidas no arquivo para se criar a base de dados com os dados que estavam nela. Se o banco de dados já estiver criado, deve verificar se nesse banco de dados contém alguma tabela ou dado já preexistente, pois se tiver, poderá gerar conflito e acabar dando erro na recuperação dos dados quando der o comando *source*.

No Cassandra, para restaurar um *keyspace* através de um *snapshot*, primeiro é necessário ter certeza que os esquemas de tabela existem, caso não existam é necessário recriá-los para que a recuperação de dados possa ocorrer. Em alguns casos é necessário truncar a tabela, depois disso localiza-se a pasta com o *snapshot* mais recente tirado, assim copia o diretório *SSTABLE* para o diretório do *keyspace* e por último roda-se o comando *nodetool refresh*.

Uma diferença entre o *snapshot* do Cassandra com o *mysqldump* do *MySQL* é que no Cassandra os dados apenas podem ser restaurados de um *snapshot* se o esquema de tabelas existir, já no *mysqldump* do *MySQL*, é gerado as instruções SQL que criam os esquemas necessários para a criação do banco de dados e para sua recuperação de dados.

4.2.4 Tabela comparativa

Por fim se construiu uma tabela buscando resumir as atividades analisadas para cada SGBD. A Tabela 4.2 resume as tarefas realizadas no Cassandra e no *MySQL*.

Tabela 4.2: Tabela comparativa das atividades para os SGBDs Cassandra e MySQL.

Atividades	Cassandra	MySQL
Modelo Conceitual	Modelagem dirigida a consulta, Gerar o diagrama entidade-relacionamento, e um fluxo de consulta.	Modelagem dirigida a dados, Gera o diagrama entidade-relacionamento
Modelo Lógico	Realiza as 5 regras propostas. Gera tabelas que se adequem as consultas.	Normalização do DER. Constroi tabelas de acordo com as entidades e relacionamentos, e atribui tipos aos atributos.
Modelo Físico	Atribui tipos aos atributos das tabelas, busca melhorar eficiência, e gera esta no SGBD por meio do CQL.	Busca melhorar eficiência e cria a tabela no SGBD por meio do SQL.
Autenticação	Realiza a autenticação comparando o nome e a senha guardados com os fornecidos pelo usuário.	Realiza a autenticação comparando o nome, senha e host guardados com os fornecidos pelo usuário.
Autorização	Oferece a funcionalidade ROLE, utilizada para facilitar controle dos usuarios.	Possui uma grande diversidade de privilegios que podem ser concedidos a um usuário.
Auditoria	Gera logs das atividades sequencialmente ou em uma tabela. Possível utilizar triggers.	Gera logs das atividades sequencialmente. Possível utilizar triggers.
Backup	Realizado por meio do comando mysqldump. Produce comandos SQL	Realizado por meio de snapshots. Possível realizar backups incrementais.
Recuperação	Realizada por meio da execução dos comandos SQL criados no backup.	Necessita que os esquemas das tabelas existam para que um keyspace seja recriado por meio do snapshot

Capítulo 5

Conclusão

Neste trabalho foi realizado uma análise comparando dois SGBDs diferentes, MySQL e Cassandra, pertencentes a dois modelos de bancos de dados diferentes, modelo relacional e NoSQL, quanto as atividades de um administrador de banco de dados. Essa análise consistiu em, para as atividades selecionadas da administração de banco de dados, estudá-las e executá-las em ambos casos para entender suas diferenças e semelhanças. Analisou-se a modelagem de dados, tratamento de segurança, monitoramento do banco de dados, *backup* e recuperação de dados.

Observou-se que por possuírem uma linguagem semelhante, o CQL e o SQL, alguns comandos realizados também são semelhantes, assim como algumas atividades, como criação de usuários e concessão de privilégios, criação de bancos de dados e tabelas, entre outras. Apesar de alguns comandos serem semelhantes, como para concessão de privilégios, a abordagem ainda assim é diferente, onde o Cassandra utiliza *roles* para facilitar a gerência dos privilégios sobre os usuários enquanto o MySQL possui uma lista de privilégios mais extensa quando comparada com a do Cassandra possibilitando um maior controle do DBA sobre os usuários. As diferenças existentes nas arquiteturas são refletidas em certas atividades do DBA, como para modelar o banco de dados, onde para o Cassandra, por possuir uma arquitetura que busca facilitar a consulta em sistemas com grande volume de dados, é necessário um estudo maior sobre a aplicação e as consultas que este irá realizar no banco de dados. A execução da modelagem do banco de dados para o Cassandra seguindo o artigo do Chebotko et al. foi interessante para um melhor entendimento das diferenças entre as arquiteturas. Os dois SGBDs possuem comandos de *backup* diferentes, porém com algumas semelhanças baseadas nas *flags* que podem ser utilizadas para determinar o que deve ser feito com o *backup*, já na parte de recuperação os dois possuem diferença quanto a necessidade da criação de esquemas, o Cassandra pede que tenha o esquema criado para que ocorra a recuperação, já com o MySQL esta estrutura é criada no processo.

É possível observar que não existe um banco de dados melhor de se administrar do que outro, mas pode se observar que para certas aplicações um possui vantagens sobre o outro. O Cassandra está mais preparado para atender uma aplicação que busca escalar e trabalhar com um grande volume de dados, para isso ele necessita de um grande esforço para modelar o banco de dados onde é modelada uma tabela para cada consulta. Já o MySQL possui um processo de modelar os dados mais conhecido e conseqüentemente mais facilitado, sendo por vezes melhor de ser utilizado em sistemas mais simples.

Os dois bancos de dados estão em constante avanço e funcionalidades novas estão sempre sendo lançadas, buscando melhorar a performance utilizando menos recurso, assim como buscando facilitar as tarefas realizadas pelo DBA. Para trabalhos futuros pode ser feito uma atualização das funcionalidades disponíveis para cada banco de dados, comparar as versões pagas que possuem diversas funcionalidades interessantes relacionadas as atividades do DBA, e comparar as atividades do DBA que não estão presentes nessa pesquisa.

Referências

- [1] Apache cassandra 3.0 documentation. vii, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
- [2] Cql for cassandra 2.2 and later, documentation. 25
- [3] The zettabyte era—trends and analysis. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html. Accessed: 2016-04-31. 1
- [4] *MySQL 5.7 Reference Manual*, jul 2016. 7, 26
- [5] Kristi L Berg, Tom Seymour, and Richa Goel. History of databases. *International Journal of Management & Information Systems (IJMIS)*, 17(1):29–36, 2012. 11
- [6] Jonas César de Sousa Pontes Bernadette Farias Lóscio, Hélio Rodrigues de Oliveira. Nosql no desenvolvimento de aplicações web colaborativas. 2011. 7, 8
- [7] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000. 7
- [8] Artem Chebotko, Andrey Kashlev, and Shiyong Lu. A big data modeling methodology for apache cassandra. In *Big Data (BigData Congress), 2015 IEEE International Congress on*, pages 238–245. IEEE, 2015. 23, 24, 28, 29, 30, 31, 32, 35
- [9] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976. 30
- [10] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 28
- [11] Ramez Elmasri, Shamkant B Navathe, Marília Guimarães Pinheiro, Claudio Cesar Canhette, Glenda Cristina Valim Melo, Claudia Vicci Amadeu, and Rinaldo de Oliveira Moraes. Sistemas de banco de dados. 2005. 4, 5, 6
- [12] Carlos Alberto Heuser. *Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS*. Bookman Editora, 2009. 13
- [13] Eben Hewitt. *Cassandra: the definitive guide*. "O'Reilly Media, Inc.", 2010. 23, 24
- [14] Maria Indrawan-Santiago. Database research: Are we at a crossroad? reflection on nosql. In *Network-Based Information Systems (NBIS), 2012 15th International Conference on*, pages 45–51. IEEE, 2012. 8, 9, 10

- [15] Karamjit Kaur and Rinkle Rani. Modeling and querying data in nosql databases. In *Big Data, 2013 IEEE International Conference on*, pages 1–7. IEEE, 2013. 10, 11
- [16] Bernadette Farias Lóscio, Hélio Rodrigues de Oliveira, and Jonas César de Sousa Pontes. Nosql no desenvolvimento de aplicações web colaborativas. *VIII Simpósio Brasileiro de Sistemas Colaborativos*, 2011. 2
- [17] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67, 2012. 1
- [18] A B M Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. *International Journal of Database Theory and Application*, 2013. 9
- [19] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013. 10
- [20] Craig Mullins. *Database administration: the complete guide to practices and procedures*. Addison-Wesley Professional, 2002. 2, 12
- [21] Jaroslav Pokorný. Nosql databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1):69–82, 2013. 7, 9, 10
- [22] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, 2008. 8
- [23] Raghu Ramakrishnan and Johannes Gehrke. *Sistemas de gerenciamento de banco de dados-3*. McGraw Hill Brasil, 2008. 5, 12
- [24] Neeraj Sharma, Liviu Perniu, Raul F Chong, Abhishek Iyer, Chaitali Nandan, Adi-Cristina Mitea, Mallarswami Nonvinkere, and Mirela Danubianu. database fundamentals. *IBM Canada*, pages 96–101, 2010. 1, 5, 6, 28, 30, 32
- [25] Tatiana Simmonds. Teaching database administration in the world of big data and small budgets. *Friday 5th July 2013 University of Sunderland*, page 45, 2013. 12
- [26] Marko Svaljek. *Cassandra Succinctly*. Syncfusion Inc., 2015. 2